TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

PABLO PASCUAL OLIVAS

SAFETY RELATED OBJECT AND PATTERN RECOGNITION FOR AUTONOMOUS MACHINES

Master's thesis

Examiners: Kalevi Huhtala and Heikki Huttunen

The examiners and topic of the thesis were approved on 31 January 2018

ABSTRACT

PABLO PASCUAL OLIVAS: SAFETY RELATED OBJECT AND PATTERN RECOGNITION FOR AUTONOMOUS MACHINES
Tampere University of Technology
Master of Science Thesis, 78 pages
February 2018
Master's Degree Programme in Information Technology
Major: Audio-Visual Signal Processing
Examiners: Kalevi Huhtala and Heikki Huttunen

Keywords: Pattern recognition, ROS, Neural networks

Automation is nowadays everywhere, from automated machines in factories that build goods, to robots cleaning your own house. Everyday new ideas about automation come out, trying to make life easier for every person and worker.

Work automation is the process where production tasks, that are usually performed by humans, are transferred to some sort of technological element. This change normally comes with optimization of the use of resources, as energy and materials, besides incrementing the quality and precision of the product, while reducing the time needed to make it.

This project's goal is to achieve safety measures during the positioning of a vehicle handling cargo, in order to pick up a demountable. To perform such process, several devices are used, a 3D sensor that calculates shape and distance of the objects in front of it using time of flight measurements and a video camera, it will be used to perform the search of the objects, for example, pedestrian detection in this project, in order to give the extra information needed for the software to adapt the movement of the vehicle until its position is the required one to hook the cargo and load it on the vehicle without taking any risk.

The main software resources that were used in the project are Matlab and C++ based Robot Operating System (ROS) codes.

PREFACE

Here is written the Master's Thesis "Safety related object and pattern recognition for autonomous machines", It is based on a project performed at Tampere University of Technology regarding vehicle automation. It has been written to fulfill the requirements needed for students of this university to graduate. I worked on this project from November 2017 to April 2018.

The project was performed by Automation and Hydraulic Engineering laboratory, where I was research assistant. My research took part in an already began project.

First of all, I would like to thank the team of the project Lauri, Mika and Antti for giving me the opportunity to work with them in this project while doing my Master's Thesis and for their guidance and ideas. Special thanks to Antti for all the help he gave me during this project time.

I want to thank my parents and my aunt, Rian, for being always there no matter what is the problem or what I need, they will do their best in any situation supporting and encouraging me even if I do not deserve it.

Thanks to Carlos for having the courage to join me in this new adventure in Finland and for been one of the best friends and flatmates one can have, even in my worst moments.

Thanks to Jose and Alberto for all the dinners, the fancy foods and the sushi, for all the laughs just chatting or playing FIFA, for letting them know that our place can be the "common room" and getting the same response from their home.

To Marina, for making me laugh every time I need and being always there if I want to talk. For all the pictures that makes us both want a puppy or a new cat, for being as crazy as she is planning trips at every second so we can meet again and be silly together.

Finally, to all the friends I have met during my university years from both, Carlos III and TUT universities, you are too many to be listed here but I really appreciate all the moments we have spent together, laughs, parties, coffees when the study became to hard and even those study hours in the library helping each other. Without all this people it would have been impossible for me to get this far.

Thank you all.

In Tampere, Finland, on February 17, 2018

Pablo Pascual Olivas

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

ROS	Robot Operating System
TOF	Time Of Flight
CMOS	Complementary Metal-Oxide-Semiconductor
APS	Active Pixel Sensor
CCD	Charge-Coupled Device
PMD	Photon Mixer Device
INS	Inertial Navegtion System
RADAR	RAdio Detection And Ranging
LIDAR	Laser Imaging Detection And Ranging
ROI	Region Of Interest
HOG	Histogram of Oriented Gradients
SVM	Support Vector Machine
CNN	Convolutional Neural Network
ACF	Aggregate Channel Features
TUT	Tampere University of Technology

θ	Phase
d	Distance
c	Speed of light
π	Number Pi
f	Frequency
I_l	Intensity level at the left of an edge
I_r	Intensity level at the left of an edge
σ	Blur scale of the edge
H_{ij}	Gaussian filter kernel
i	Horizontal position in kernel matrix
j	Vertical position in kernel matrix
k	Kernel size defining variable
σ_g	Variance
G	Gradient
G_x	Gradient in horizontal axis
G_y	Gradient in vertical axis
θ_e	Angle of the edge direction
y_1	Horizontal axis in Pinhole image plane
y_2	Vertical axis in Pinhole image plane
x_1	X axis in Pinhole 3D space
x_2	Y axis in Pinhole 3D space
x_3	Z axis in Pinhole 3D space
f_p	Focal length in Pinhole model

1. INTRODUCTION

During this chapter, the most relevant topics regarding this project will be analyzed. For example, the main focus of the project and the motivation to complete it. After such introduction, the structure of the thesis will be defined in order for the reader to understand more precisely the contents of each chapter.

1.1 Motivation of the project

Since the first invention of man, humans have been developing new technologies and methods to perform in an easier way the task that they needed to complete. Since the industrial revolution, machines have been part of our daily work, letting us get much better results and resource saving while needing less time to finish the duty.

Society is now in the information era, this means that we can communicate with each other and with machines in such short times they can be neglected when we talk about the task process. Machines can communicate between themselves and perform operations in less time than a millisecond, this makes tedious calculations and processes that required much longer times for humans to complete them, to not being even taken into account when planning the project.

The first time someone thought about an autonomous vehicle was in 1939, the idea came from Norman Bel Geddes, after that, in 1980 Mercedes-Benz created the first autonomous van guided by vision. During the last years, autonomous vehicles have been developed further, now vehicles not only can drive themselves, but they can even detect accidents that other vehicles will have and warn the people inside the car.

A lot of companies are interested in those self driven vehicles and the development of software related to them, as some of those vehicles are already being sold to regular users, a new market is starting to become available, so further software development could be very convenient in a short period of time, this will be the main topic of this project.

1.2 Objectives

The main idea of this project is to develop software that will perform the safety measures while driving a cargo vehicle in front of a demountable, until the position needed to hook the cargo is reached. Such measures will be related with pedestrian detection and position check, for the vehicle to know if such pedestrian could be positioned in a dangerous place, if not, it will continue the approach to the demountable.

Checking the position of other obstacles in the scene to avoid problems when approaching the demountable position is also performed.

Finally, detection of warning signs that might be placed in the demountable as flammable, explosive or biohazard, in order for the vehicle to pick them up accordingly to the safety conditions.

Such process will be carried out using several devices that will communicate between them, such as the 3D sensors, the camera and the computer performing the calculations.

The communication technique will start when projecting infrared light from the back part of the vehicle, this is done to detect shapes and objects on its way, that light will be recorded by a 3D sensor that will send the data to a computer.

The computer will create a virtual environment with the position of all the objects recorded, as well as the vehicle and sensors in a 3 dimensional space. With that information, the computer can start making calculations regarding the positioning of the vehicle with respect to its environment, but some extra knowledge is needed.

The camera will record in real time the image behind the vehicle. Then it will be checked, whether or not, the vehicle can still approach the demountable, in order for it to be placed in the perfect position to stop the vehicle in front of the demountable. This will be achieved using object recognition mechanisms, through image processing techniques.

All that process will be done taking into account possible obstacles surrounding the demountable as, for example, people walking by or objects placed between the demountable and the vehicle. Also warning signs detection will be handled, in order for the driver or vehicle to know when to have extra care when taking some cargo.

The main objectives of the project are:

- Receive data in real time from the time of flight device, to record infrared light.
- Process the recorded infrared light in a point cloud, in order to recreate a 3 dimensional space with the objects recorded, as well as the position of sensors and vehicle.
- Receive information from a video camera and process the image in real time, in order to detect pedestrians in the environment.
- Check in real time the safety measures to avoid obstacles, possible persons in the way and detect warning sign conditions, as flamable or explosive.
- Proximity warning while moving the vehicle.
- Recognize the company logo that might be in the demountable.
- Perform all the calculations needed to adapt the position of the vehicle, taking into account the data obtained.

1.3 Contents of the project

During this section, all the chapters that appear in this document will be reviewed briefly, for the reader to have a simple explanation about what he or she is going to find. This document is divided into five chapters.

- Chapter 1, introduction. This chapter will give the introduction of the project. The motivation of the project and the objectives are examined. A brief explanation of the method used to obtain those results is also given. Finally, the contents of the document are listed.
- Chapter 2, theoretical background. Inside this chapter, there will be given the theoretical background regarding the project, from the point of view of all technologies and software used on it. The first part will talk about Time Of Flight sensors, as this is the main device used in the project, for detecting the distance to the objects. Robot Operating System (ROS) will be also explained, as it is the main set of libraries and tools used for the developing of the project, all devices and software are connected through it. Some basic notions of Object recognition, with the main methods and how they work is also written in this chapter. Autonomous vehicles section will explain the basics of the technology, software and hardware that is presented on them.
- Chapter 3, research methodology and materials. During this chapter the reader will find a list of the materials used during the project, followed by the basic information of the devices used during the project, to capture video and distance information during the project. The next part is an explanation of the methodology used during the project. It will be explained step by step, everything that was needed and achieved during the implementation of the work, from the start of the recording, until the obtained results, going through all the software developed, the methods used and the steps taken in them for the pre-processing, calculations of information and object detection. Also some explanations about the tools used will be placed inside this chapter.
- Chapter 4, results and analysis. During this chapter, the final results obtained from each test during the project will be exposed, such experiments will be regarding the different parts of the project, as pedestrian detection and warning, obstacle warning or warning sign detection taking into account the different environments, where the data used was recorded.
- Chapter 5, conclusion. In this chapter, the final ideas gathered during the project will be interpreted, in order for the reader to understand how was the final stage on the project and what kind of information did the team arrange after the implementation of each test stage. This information will determine which is the best approach to the problem and future lines of work that will help to develop even further the project.

2. THEORETICAL BACKGROUND

2.1 Time of flight (TOF) sensors

Time of flight is a technique used to calculate distance between objects, using infrared light measuring the time taken for the light to go from the emitter to the object and return back to the sensor.

The sensor used in this project have a camera lens and a Complementary Metal-Oxide-Semiconductor (CMOS) sensor, also called Active Pixel Sensor (APS). This sensor allow current technology to introduce extra features as luminosity control, contrast correction and analog-digital conversion in a single chip. [7]

Each pixel inside the CMOS sensor has its own photo diode, it creates an electric current depending on the light intensity received. The CMOS sensor has filters, those will allow to get in only red light, green or blue.

CMOS technology is based on using a group of pMOS and nMOS transistors together. Such group of transistors makes that, in state of rest, the only energy consumption is due to parasitic currents, this causes a very low energy utilization. One extra characteristic, is that CMOS circuits can regenerate the signal, even if it degraded, if the signal is still between the noise levels allowed by the circuit, this means that the output of the CMOS will be 0 or 1 depending on the input signal.

In figure 2.1, the reader can see how CMOS inverter looks like as an electronic circuit.

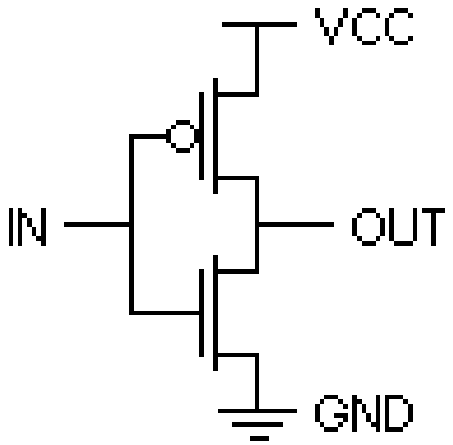

Figure 2.1. *CMOS inverter circuit. (Schaltkreis)*

Some of the advantages of CMOS technology are:

- Low energy consumption.
- Is cheaper than similar technologies, as Charge-Coupled Device (CCD).
- Simultaneous handling of pixel information.
- Analog-digital converter integrated in the chip.
- (Almost) no Blooming effect.
- Faster visualization of data.
- Higher image frequency than in similar technologies as CCD.

This TOF sensor can use different methods to calculate the distance to an object:

- Basic measurement: this method just take into account the time it took for the beam of light to go from the emitter to the object and reach back the detector, the calculation of the distance is very simple knowing the speed of light. This procedure can make a 200m measurement with 6mm error.
- Phase-shift: Is the most used technique nowadays. A periodic signal is needed to be emitted. As the light will cover the distance to the object twice, it can be computed from the phase-shift (θ) as in equation 2.1. [8]

$$d = \frac{c}{4\Pi f}\theta \tag{2.1}$$

The device used on this project is a Photon Mixer Device (PMD) camera, this kind of sensors are capable of calculate both, the distance of each pixel to an object and the intensity of the light. This camera applies CMOS technology with Phase-shift methods for the depth measurement.

2.2 Robot Operating System

"The Robot Operating System is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms." [9]

ROS can offer services developed for computer clusters, formed from several different machines, some of those service are: [10]

- hardware abstraction.
- control over devices connected to the machine.
- commonly used functionality.
- inter-process messaging.
- package management.

ROS file system is based on packages and manifests. Packages are the software organization unit of ROS code, containing inside them the needed libraries, .exe files or scripts to run the desired program properly.

Manifests are the description of the package, it sets the conection between packages. Some of the tools that can be used when working with the file system are, rospack to get information about the package and roscd to go to the folder of an specific package. [11]

During the project, ROS catkin package system was used, this catkin system is the official build system for ROS, catkin combines CMake macros and Python scripts, this gives extra functionalities on top of Cmakes's workflow. Catking allows better distribution of packages, cross-compiling support and portability. [12]

ROS uses several tools to have the programmed codes working, one of the most important tools are nodes. Nodes in ROS are processes that makes some kind of computations. Nodes communicate with each other using streaming topics, WPC services and Parameter Server [13]. Basically, each node represent one of the parts of the project, in this case, some of our nodes would be the infrared emitter with 3D sensor and camera.

Nodes communicate with each other using topics, the way they do so is by publishing messages inside the topic, in order for other node to subscribe to it and read the messages already published inside, those messages are stream of data. Apart from that, there are files of text that will indicate the data structure of the messages, using message definitions. [14]

The topics, through which messages are sent, are basically buses ready for the nodes to send information between them. Nodes does not know to which other node they will be communicating with, only that they have to publish the information to a specific topic and whatever comes next, is the duty of the topic or the node that must get that information. There can be several publishers and subscribers (readers) of a topic. Topics are unidirectional communication systems. [15]

For ROS, a bag is a file format that will store message data. The information that is saved inside a bag is usually saved in a reading loop from a topic, every time the topic receives new information, the recording will read it from the topic and store it in the .bag file. In order to avoid problems with the time stamp of the files, rosbag tool includes an option to simulate a clock when playing recorded data, when reading again that data, ROS can be specified to use the required time stamp and clock from it. [16]

The most important part when using ROS is launching roscore, without this command used on the Linux terminal at the beginning of the working session, ROS will not be running. Roscore is a set of nodes and programs that are required for a ROS-based system, without it, nodes can not send messages to each other. Roscore starts a ROS master, ROS Parameter Server and rosout logging node. [17]

One useful tool inside ROS to understand how your nodes and topics are working, is rqt_graph tool. rqt_graph creates a graph that shows how the system is behaving at every moment, all the existing nodes and the topics through which they communicate in which direction. This is very useful when the programmer is trying to detect errors, or when trying to understand programs already made, as the connections inside those programs might not be known. An example of the output from this command can be found in figure 2.2.

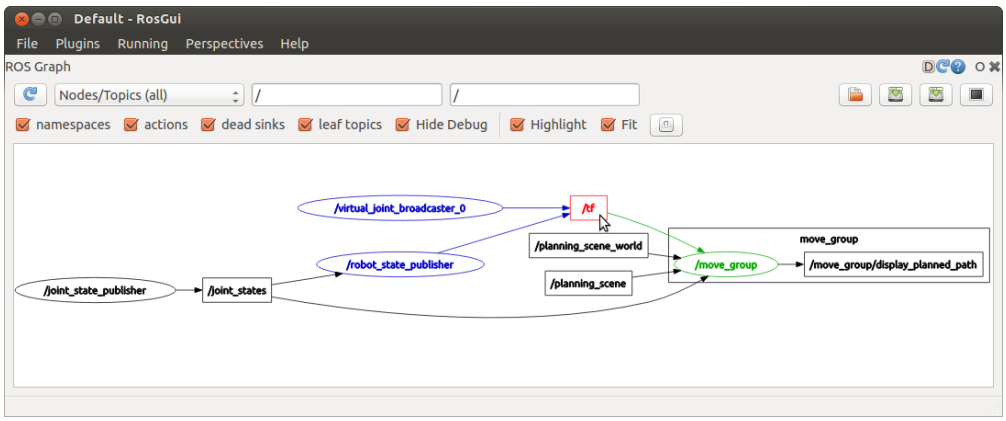

Figure 2.2. *Example of rqt_graph output. [1]*

One of the tools used during every test and to check the results and data obtained during this project, was Rviz tool. Rviz is a visualization tool used in the project to examine sensor information and tf (transform of coordinates) frames. This tool creates a 3D space in which it will place all the data published by our nodes, in our case devices, as transform, point cloud, pose, etc.; for example, in our project, whenever the vehicle moves, also all the devices attached to it move. Rviz will show the movement of the point cloud and, because of it, how those devices move with respect of the objects behind the vehicle.

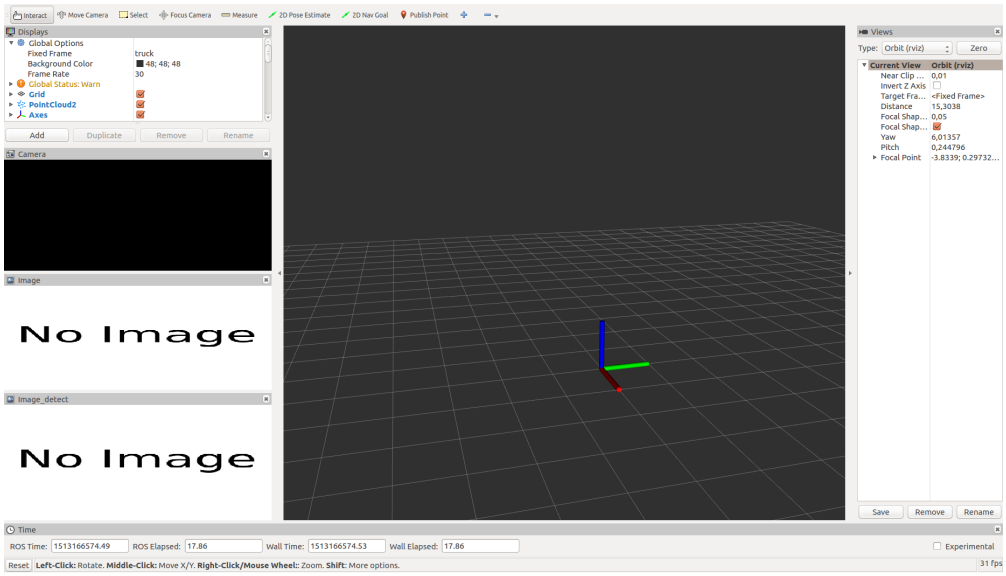

Figure 2.3. *Rviz window configuration for the project.*

As seen in figure 2.3, on the left part, the configuration visibility and options for possible displays during the use of Rviz are shown; below that, the "camera" window which will show the image recorded by the video camera, adding on top of it our 3D data, point cloud, ground plane and demountable marker. The "image" window, will show the image that is read from the .bag file of data stored in the computer, depending on the needs of the team, the region of interest or the whole image will be displayed with, for example, symmetry calculations or Hough circles transform. Finally, the last window appearing on the left is "image_detect", here, the object and pedestrian detection will be shown to the user in real time, with their corresponding warnings (red rectangles, stop and pedestrian).

On the central part, it can be seen the 3D space created by Rviz in which all our nodes (devices) appear as axis coordinates, also, is possible to see the point cloud created with the data from the 3D sensor. This point cloud is the representation from the objects behind the vehicle, to which the vehicle is approximating.

At last, on the right part of the image, information about the view point, where and how the 3D view is shown is presented to the user.

ROS framework is used during the whole development of the project to analyze the position of each device as well as the vehicle with respect to its environment. Using ROS C++ based software and merging it with Matlab software, the team was able to develop all the functions required for the completion of the goals.

2.3 Object recognition

The goal of object recognition is to find and identify objects that appear inside an image or a video sequence. The approach to this problem used during the project was using, for example, HOG + SVM for pedestrian detection or to train neural networks for the warning sign recognition tasks.

2.3.1 Edge detection algorithm

Edge detection algorithms are based on the change of brightness between pixes in the image, those changes usually identify discontinuities in the object appearing in the image, those changes that can translate, for example, in the edges of the object. [18]

The typical meanings of the detection of an edge in machine vision are: [19]

- Discontinuities in depth.
- Discontinuities in the direction of surface normals.
- Differences in surface state.
- Deviation of image brightness

Having the edges inside an image detected will mean less computational power needed and faster way to process the data, as most of the image will be discarded during this procedure. In the cases where this project is intended to be used, this detection algorithm will translate the edges into the actual shape of the object recorded,in this case all the recorded image.

During this procedure, a threshold to define where edges can be found is needed in order for the program to avoid false edges in some cases, as change of light characteristics in the same surface.

When detecting edges in real images step edges are not the usual case, some effects will affect the edges and will make this detection task more difficult. Some of those effects are: [20]

- Focal blur, it becomes a problem as real cameras have a finite depth of field and point spread functions.
- Penumbral blur, this effect is caused because of lights that have a radius different from zero, affecting how the shadows would be comparing them to the ideal case.
- Shading that can appear on a smooth surface, giving the illusion of a different face in the object.

One way to try to avoid such errors could be using a Gaussian smoothed step edge as pre-processing, this will simulate the ideal step, erasing more or less the blurring effects that might appear in the image. [2]

To model an image having at x = 0 only one edge, equation 2.2 could be used:

$$f(x) = \frac{I_r - I_l}{2}\left(erf\left(\frac{x}{\sqrt{2}\sigma}\right) + 1\right) + I_l \tag{2.2}$$

An adjust on σ must be performed depending on the quality of the image, this adjustment will determine whether or not a true edge is considered as so or not.

The main problem for a machine to detect edges inside an image is that, sometimes, the edge is not easy to detect, as there might be a transition on the brightness of the image during several pixels, in this case, the machine might not know the exact position of the edge or if there is an edge at all. An example of such transition in pixels, that may lead to a mistake in the edge detection is shown in figure 2.4.

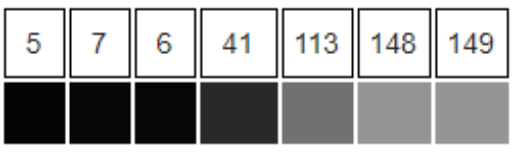

***Figure 2.4**. Brightness transition between consecutive pixels example. [2]*

The most used methods for edge detection are based mainly on two different mathematical methods, search-based and zero-crossing based. Search-based methods calculate the edge strength by using first order derivatives, with that information, the local directional maxima of the gradient magnitude will be computed taking into account an approximation of the edge orientation. In the zero-crossing based approach, zero-crossings will be computed using the second order derivatives, those derivatives are calculated using the Laplacian or from a non-linear differential expression. With that information, the next step is the calculation of the edges. [21]

John Canny created the method of one of the best edge detectors, this method depends on a multi-stage algorithm. This method have some characteristics as low error rate, edge point detected is localized on the center of the edge.

The process used for Canny detector is the following: [22]

- Smooth step Gaussian filter applied to the image. This filtering method is performed as noise can alter a lot the output from the edge detection algorithm, the formula for the Gaussian smooth step filter kernel of size (2k+1)x(2k+1) is given in equation 2.3

$$H_{ij} = \frac{1}{2\pi\sigma_g^2} exp\left(- \frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma_g^2} \right); 1 \leq i, j \leq (2k+1) \quad (2.3)$$

- Calculation of the intensity gradients inside the image. This operation will apply the gradient operator to the edge in both, horizontal and vertical directions, leading to the calculation of the gradient and direction of itself. The formulas to obtain the gradient and the angle of the edge are shown in equations 2.4 and 2.5 respectively.

$$G = \sqrt{G_x^2 + G_y^2} \quad (2.4)$$

$$\theta_e = atan2(G_y, G_x) \quad (2.5)$$

- Non-maximum suppression is applied, this will eliminate false edges. This process will only keep the part of the edge where the biggest intensity change appear, cleaning in this way the final output of the process, giving sharper edges.
- Double threshold to find possible edges is applied. This process will apply two thresholds to the output from the previous step. If an edge has bigger gradient value than the higher threshold, it will be considered strong, while if its gradient value is between both thresholds, it, will be considered weak, edges with lower values are discarded.
- Hysteresis tracking from the edges detected. After all the edges are already classified between weak and strong, the weak edges will go through the final process, the algorithm will check each weak edge, in order to find out if it is connected to a strong edge or not, analyzing its 8 neighbor pixels, if the edge is connected, it will be considered valid, if not, the most probable case is that it was created because of some artifact and it will be considered not strong enough.
- Deletion of the edges considered not strong enough, this will finish the algorithm with the best edges obtained from the original image.

There are several methods used for object recognition that might be based on edge detection, but some others are not. Some of those methods are:

- Primal sketch, this approximation to object recognition is based on the derivation of the main components from an image, as edges or regions; it receives such name because, the idea of this method, is to have a pencil sketch like result.
- Generalized cylinders, This method is based on recognition of parts inside an image, what this method tries to calculate is the supposed position to place a cylindrical mesh to get the most similar reconstruction of the required object in the scene.
- Geons, This method is based on identifying geons, or basic geometric figures, inside the image, one example to understand this concept is an ice cream, the shape of it on an image could be a circle as the ice cream and an inverted cone as the cone. The main problem comes when several objects can be constructed from the same geons, for example, a person with a winter hat could be consider with a circle as the head and a cone as the hat. Because of this, when using this method is difficult for the machine to compute the closest object, as geons should be viewpoint invariant. [23]
- Appearance-based methods, this approach to object recognition is based on using pre-selected images as template from specific objects. Having more example images will translate into more success rate at the time of detecting the desired object.
- Edge matching, this method is based on edge detection, the process is to detect the edges inside the image, for example, using Canny edge detector and compare the result with an edge detected template of the object that the machine should recognize.

2.3.2 KD trees

KD tree is the abbreviation for K-Dimensional tree. It is a data structure that will store data inside a K-dimensional Euclidean space. The main characteristic of a KD tree is that, only perpendicular planes to one coordinate axis are used in order to subdivide the tree in nodes.

The most common algorithm to create a KD tree is:

- Each subdivision will be done perpendicular to the following axis, being them in X, Y, Z order. For example, if the subdivision of the root node was perpendicular to the X axis, the division of the child will be perpendicular to Y axis and the division of the niece on Z axis.
- The data point selected to create the division will be the median of the points inside the node, this will create a tree in which all the leaf nodes are at the same distance to the root node.

The steps followed to find a point in the tree are the following:

- Recursive search down the tree, selecting left child node if the searched point is lower than the point used for the division of the node, or right node if it is higher.
- If leaf node reached, mark it as current best.
- check each node from leaf to root, in case it could have lower distance than the current best node to the desired point. Also, check points inside the other divided node closer than the best distance known, as they might be closer points in that node.

An example of a divided 3D tree can be seen in figure 2.5. [24]

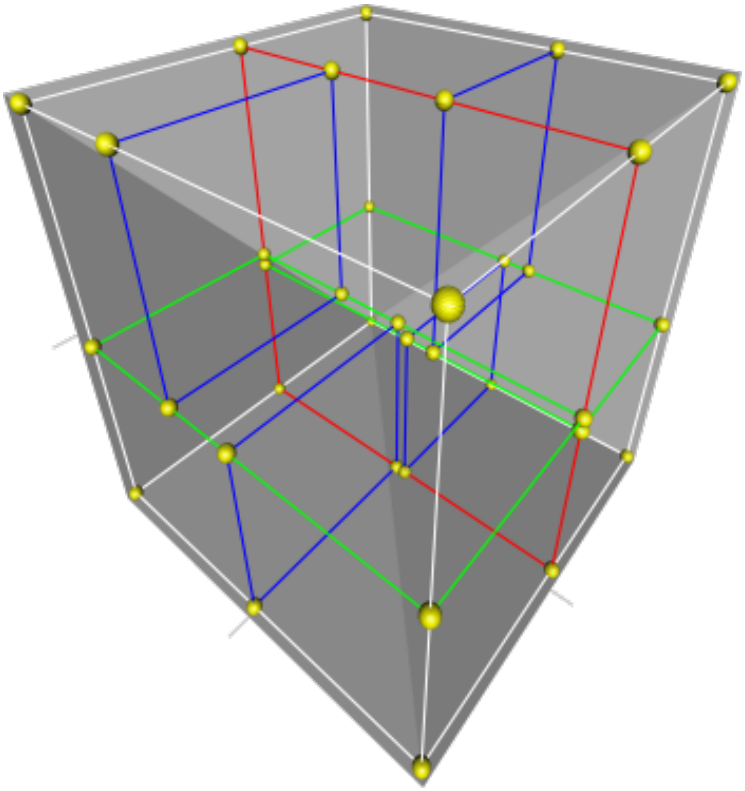

Figure 2.5. *Example of three dimensional tree with three subdivisions. [3]*

2.3.3 Pinhole camera model

This model creates the relation between a point inside a 3D space and its projection into a 2D image from an ideal Pinhole camera. In this model, the camera aperture is a single point and there is no lenses that could create distortions or artifacts in the final image.

The projection of the point observed in the 3D space will appear in the image plane, as the point in which the line joining that point in 3D space and the coordinate origin cuts the image plane. To understand this concept, figure 2.6 shows in 2D how this method works.

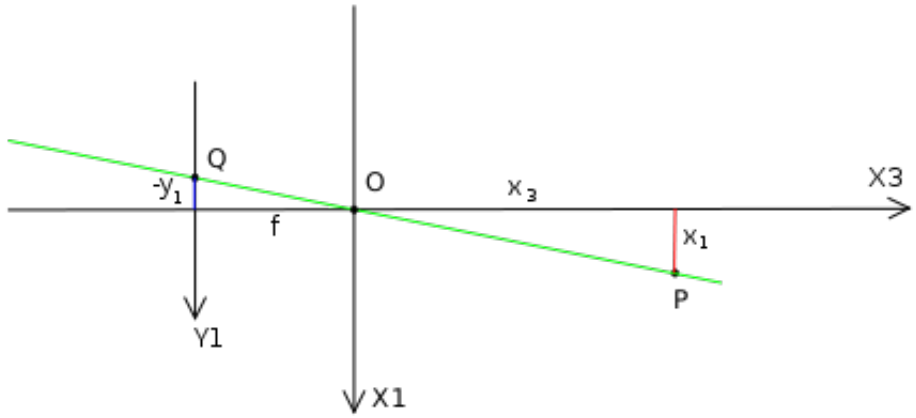

Figure 2.6. *2D Pinhole camera model example. (KYN, 23-07-2007)*

From the image model, and taking into account also the vertical coordinates that cannot be seen in it, equation 2.6 can be derived.

$$[y_1, y_2] = -\frac{f_p}{x_3}[x_1, x_2] \tag{2.6}$$

From equation 2.6, it is acquired the formula relating the position of the point in the 3D space, with the position of the same point inside the 2D image plane in which it is captured. Having this equation, knowing the position of any point inside one of those spaces, the position in the other one is easily calculated. [25]

2.3.4 Histogram of Oriented Gradients

The Histogram of Oriented Gradients (HOG), is a descriptor from the visual features appearing in an image or set of images (videos). It is used to perform object recognition inside such images. This method calculates the gradients orientations in windows of the image, the smaller the window, the more gradients will be calculated and higher resolution of the shape of the objects appearing in the image will be given to the detector. This method is also adapted to use overlapping local contrast normalization.

The idea of HOG is that, the shape of the objects appearing in an image can be characterized using the intensity gradients appearing in it. To obtain them, the image is separated in windows; for each window, a histogram of gradient directions is computed, those values are normalized afterwards depending on the intensity in that part of the image. [26]

Computing the HOG starts calculating the gradients applying a one dimensional point discrete derivative mask in one or both directions (vertical and/or horizontal), as this method is the one that has given the best results in previous research.

Once the gradients are computed, the next step is to perform orientation binning, this means that the histograms will be created in each window based on the gradient calculations computed in the previous step.

After the histograms are stored, the descriptor blocks can be created, several windows are used together to normalize their gradients depending on the intensity levels of that part of the image, the descriptor is formed by the connection from all the normalized gradients from each window in that region of the image. Overlapping appear in this step, this will make each gradient to give extra information to the final descriptor.

Such process can be seen in figure 2.7.

***Figure 2.7**. Original image with output from computation of histogram of gradients overlayed on it.*

Once the process is completed, the following step would be to train a classifier to be able to detect such shapes inside a test image, in the case of this project, real time video.

2.3.5 Support Vector Machines

Support Vector Machines (SVMs) are used in supervised training in machine learning. They are used in this project to train a model that can detect and classify objects inside an image. To achieve such goal, a set of training images are given to the HOG computer function. After the computation is ready, the outputs are given to the SVM for it to be trained and be able to detect objects inside the new (test) images given to it. [27]

To create the model, the SVM creates a space in which the classes will be positioned as points after training, that space will be divided by a hyperplane that will be the division between two classes, called the support vector. Depending on where the point of the new (test) recognized objects belongs, they will be labeled as one of the classes in the space.

The selected hyperplane in an SVM is the one giving the maximum separation between the classes. SVM algorithms are part of the linear classifiers. An example of such separation can be seen in figure 2.8.

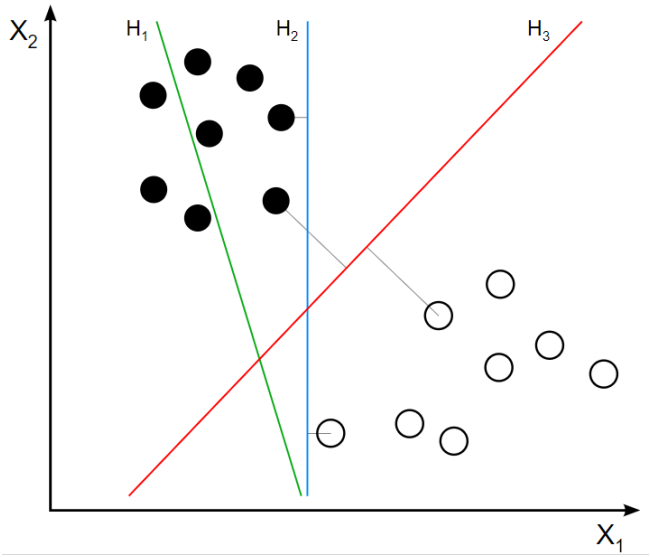

Figure 2.8. *Possible hyperplanes in an SVM, H3 giving the biggest distance between sets. (ZackWeinberg)*

In the ideal case, a SVM should separate all data into two completely differentiable categories. The problem is that, in real applications, that perfect separation is not always possible and if a close approach is obtained, that model cannot be used for new data. This is known as overfitting. To avoid such thing, SVMs use have implemented a soft margin parameter, this allows some errors during the classification while penalizing them at the same time.

As in most real world cases, the separation between sets can not be a straight line because of more than two classes in the data set, more than two predicting variables or data sets that can not be completely separated. Kernel function is used to solve that problem, this function will project the data into a higher dimensional space, this will increase the computational power of the SVM. There are several kernel functions used in SVMs as polynomial-homogeneous, perceptron (figure 2.9), sigmoid or Gaussian radial basis functions.

For pedestrian detection, a pre-trained HOG + linear SVM model in OpenCV is used in real time, analyzing frame by frame the input images recorded by the video camera. Finally, for sign detection, a R-CNN (Convolutional Neural Network) is trained using several images of each category as flammable or explosive signs. This trained neural network will be used to classify images taken afterwards and check if in such image those signs appear. Also this method is used for company logo classification.

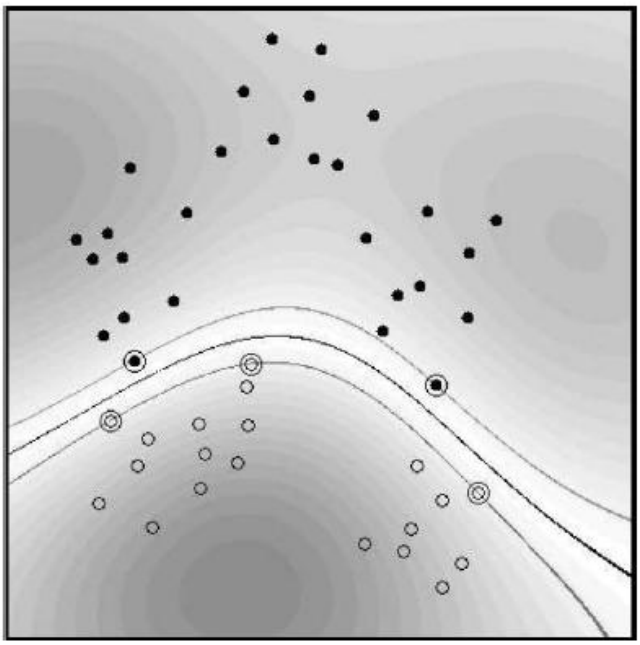

Figure 2.9. *SVM separation using perceptron kernel.(Elisfm)*

2.4 Autonomous vehicles

An autonomous vehicle, also known as self-driven car, is a vehicle capable of replicate human capacities regarding sensing the surroundings and act accordingly to them without human input. [28]

Some of the main ways for making the car feel his surroundings are via radar, lidar and computer vision. Those systems will obtain the data around the car, (like obstacles and traffic signs) that data will be processed for the machine to know its next movement as it updates the route to know exactly where the car is. Autonomous cars nowadays can use path planning methods to calculate a route.

2.4.1 Software components

This kind of vehicles need of several real time systems in order to observe and analyze all the unpredictable traffic in a road. Those systems must work with each other, this way the vehicle has all the information possible about its surroundings at any time, this will increase security. Some of the systems could be localization, surroundings perception, planning and control. [28]

For programming such vehicles, machine learning algorithms are used, they are trained with images of different situations that could happen when the car is driving. Using this method, the computer will generalize situations instead of programming every single possible outcome. This learning stage is performed several times until the computer is able to know correctly what are the situations in the images and how to act subsequently to them. [29]

2.4.2 Hardware components

For the car to be able to get all the possible information, several hardware components are usually build on it, some of them are a GPS unit, an Inertial Navigation System (INS), laser meters, RAdio Detection And Ranging (RADAR), Laser Imaging Detection And Rangig (LIDAR) and video cameras [28]. All the data obtained from those sensors is filtered and combined to update the knowledge of the environment near the car, this will update the surroundings map to avoid obstacles.

To create the surroundings map, the lasers and cameras are used in order to recreate a 3D model of the surroundings where the car can locate itself. For the localization task, the GPS and INS are used. The INS is a fusion with GPS positioning system, in case the GPS does not have accurate positioning, both acquired data will be combined using, for example, a Kalman filter. This system will estimate position, orientation and velocity of the car using accelerometers and rotation sensors. Instead of a GPS, GNSS can be used, given in most cases better results when the vehicle is moving and planning the path to follow.

Lidars are one of the main ways for self-driven cars to detect the surroundings, the lidar sends laser pulses to detect the distance to the objects, also, the speed of the objects on which the laser is reflected. When receiving the reflected laser, the lidar system will send the raw data to a program, this program will create a point cloud of the neighborhood space of the car. One of the ways the lasers are emitted, is having a 360 degrees rotation device in order to acquire all 360 degrees point cloud of the surroundings of the car but some lidars have a more narrow view. That point cloud is updated in real time for the car to know all possible obstacles at any moment [30]. An example of how a lidar represents the surroundings is shown in figure 2.10.

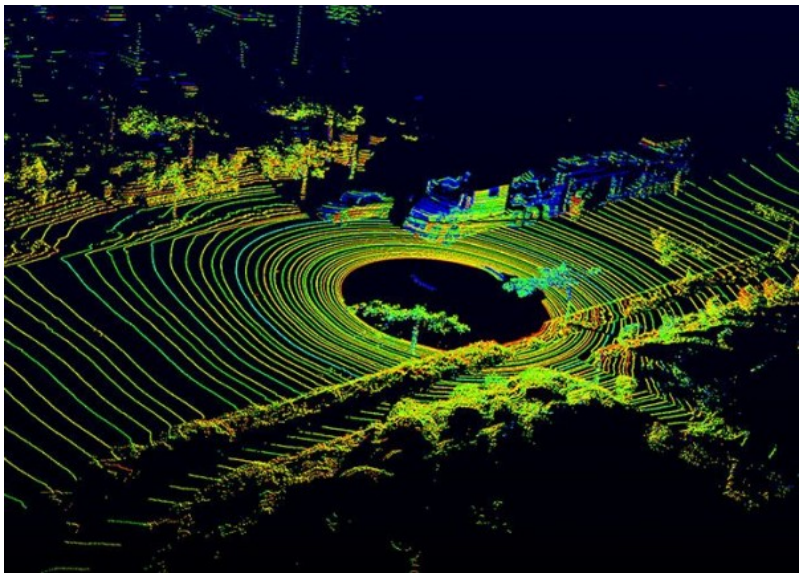

Figure 2.10. *LIDAR point cloud example. (VELODYNE LIDAR)*

Radars are the main competitor against lidars when talking about autonomous vehicles, the principle is more or less the same in both, but when lidars send infrared light and wait for the reception, radars send a radio wave at an specific frequency and awaits for it to be reflected and come back. The received radio signal will tell the car, after the appropriate calculations, where the obstacles are and which velocity they have.

The main advantages of radar is that is cheaper than lidar, even though Solid State lidars are emerging and cost less money than the previous versions. Another advantage is that adverse conditions, as snow or rain, do not affect so heavily to radar than to lidar. But the main disadvantage is that RADAR is not as accurate as lidar at the time of determining the shape of the objects. [31]

The main advantages of a Solid State lidar with respect to previous versions are the price, reliability, size and complexity issues due to mechanical problems, as this version has no mechanical components. The only problem of this kind of lidars, is that they have less field of view. [32]

Computer vision tries to acquire, process, analyze and comprehend real world images, in order to create information that a machine could understand. One of the main problems when recording such real world images is the artifacts that will appear on them, those artifacts might create false information, this will create mistakes at the time when the machine should understand the information.

Some of the most common artifacts that appear in real world images are: [33]

- Salt and pepper noise, this noise will make some pixels to tend to white or black colors.
- Uniform noise, this noise affects the whole image making all the pixels to tend to white or black color in a uniform way, through all the image.
- Gaussian noise, this noise is similar to uniform noise, but the change in the colors is not that noticeable, pixels will tend to gray colors instead of black and white.
- Viewpoint, depending on the viewpoint of an object, the image will be different from other images used to train the machine, with enough images from different points of view this problem could be solved, also, the light conditions will affect in the same way.
- Occlusion, If the object that the machine is trying to identify is partially occluded by other object located in front of it, the machine might not be able to understand the shape of the object properly and will discard it as a match.
- Scale, depending on how the image was taken, one object might appear bigger or smaller than the ones used for training, the machine should be able to recognize the object without taking into account the size of it.

- Deformations, if an object is deformed because of the way the image was taken or due to errors in the capture device or in the object, the machine should have some error threshold at the moment it has to decide, whether or not, the object is a match in order to avoid such errors in the information obtained from the image.

All noises could be avoided applying the corresponding filters to the image.

Machine learning is usually used for this kind of object detection in images. There are two main methods of machine learning, supervised and unsupervised. In the first one, the machine will be given a set of images that will have the data already labeled, the machine will then separate the data into training and test data, training data will create an inferred function, once the function is created it will be used to determine the label of the new images from the test data and will return the results, for humans to understand how good was the machine matching the images and labels. At some point, the machine can work with new images without label [34]. On the other hand, unsupervised learning will try to detect information in the structures appearing inside the images, those images will not be labeled and the machine will learn how to differentiate them by itself. [35]

3. RESEARCH METHODOLOGY AND MATERIALS

3.1 Materials used

The materials used during this project where:

- Vehicle, truck 10 meters long.
- 3D sensor O3M250.
- Illumination Unit O3M950.
- UI-5260CP Rev. 2 camera.
- Lenovo computer ThinkPad ID:4236WUL. Intel Core i5 (2nd Gen) 2520M / 2.5 GHz, 4 GB RAM.
- ROS kinetic Kame distribution.
- Linux Ubuntu 16.04 distribution.
- Matlab R2017b (9.3.0.713579)
- QtCreator 4.5.0
- C++ and Matlab libraries.

3.1.1 IR-Illumination unit O3M950 (IFM)

The illumination unit is the first step in order to detect the distance to the objects behind the vehicle, this unit will emit infrared light from the back of the vehicle and the 3D sensor will record the reflection of that light, in order to calculate the distance of each pixel.

Some of the main features that can be found in this device are his compact and robust housing, its wide operating temperature range and that it can be used in outdoors conditions with good performance, all this will help in the environments in which the project is meant to take part in.

During the first test performed, this device was placed on the top back part of the vehicle tilted down 15 degrees, this made the device to have a wider area to emit infrared light from which part of the light could be reflected. During the second test, the device was placed on the back bumper of the vehicle with minimum or no tilt, in this position, the detection of the demountable was faster as the emitter and sensor where closer to it, giving the computer more data to process. Even though less data about the ground could be recorded in this way, the team thought that with the data regarding the position of the demountable should be enough.

The Illumination unit can be seen in figure 3.1.

Figure 3.1. *Illumination unit O3M950. [4]*

Some of the characteristics of the illumination unit are shown in table 3.1.

Table 3.1. *Technical details of the O3M950 IR-Illumination unit. [4]*

Electrical data	
Operating voltage [V]	9...32 DC
Current consumption [mA]	<5000
Power consumption [W]	45
Protection class	III
Type of light	infrared light
Wave length [nm]	850
Range	
Angle of aperture	70 x 23
Image repetition frequency 3D [Hz]	25 / 33 / 50
Software / programming	
Parameter setting options	Via PC and O3M15x / O3M25x with ifm Vision Assistant
Interfaces	
Communication interface	MCI
Operating conditions	
Ambient temperature [°C]	-40...85
Note on ambient temperature	with high image repetition frequency of 25 Hz
Storage temperature [°C]	-40...105
Protection	IP 67; IP 69K;,(with mounted connectors or protective caps)
Mechanical data	
Weight [g]	1270
Dimensions [mm]	143 x 85 x 83.5
Materials	diecast aluminium

3.1.2 O3M250 3D sensor (IFM)

This is the sensor used during the project to obtain 3D information from the back of the vehicle, it gives us reliable 3D detection of scenes and objects using time-of-flight technology.

Some of the main features for which this device is useful are its compact size, robustness and the time-synchronous output of a 3D point cloud while streaming live video data. The sensor can be seen in figure 3.2.

Figure 3.2. *3D sensor O3M250. [5]*

The TOF devices already mounted in the vehicle can be seen in figure 3.3.

Figure 3.3. *Illumination unit O3M950 and 3D sensor O3M250 mounted on vehicle.*

Some of the characteristics of the sensor are shown in table 3.2.

***Table 3.2**. Technical details of the O3M250 sensor. [5]*

Application	output of 3D image data; output of 2D image data
Electrical data	
Operating voltage [V]	9...32 DC
Current consumption [mA]	<500
Power consumption [W]	4.5
Protection class	III
Range	
Image resolution [pixels]	640 x 480
image resolution 3D [pixels]	64 x 16
Angle of aperture cylindrical [°]	90
Angle of aperture	90
Angle of aperture 3D [°]	70 x 23
Image repetition frequency [Hz]	25
Image repetition frequency 3D [Hz]	25 / 33 / 50
Software / programming	
Parameter setting options	via PC with ifm Vision Assistant
Interfaces	
Communication interface	CAN; Ethernet
Number of CAN interfaces	1
CAN	
Transmission rate	250 (125...1000) kBaud
Protocol	CANopen; UDS
Ethernet	
Number of Ethernet interfaces	1
Protocol	UDP/IP
Factory settings	IP address: 192.168.1.1,subnet mask: 255.255.255.0
Operating conditions	
Ambient temperature [°C]	-40...85
Note on ambient temperature	with high image repetition frequency of 25 Hz
Storage temperature [°C]	-40...105
Protection	IP 67; IP 69K;,(with mounted connectors or protective caps)
Max. immunity to extraneous light [klx]	120
Mechanical data	
Weight [g]	1067.6
Dimensions [mm]	143.8 x 70.1 x 85
Materials	diecast aluminium
Sensorart	PMD 3D ToF-Chip / 2D Chip

3.1.3 UI-5260CP Rev. 2 camera (IDS)

This camera, from the company IDS, is the main source of real time video recording from the back of the vehicle. Some of the main features of this camera are better light sensitivity, allowing to work with low-light conditions, dynamic range and color reproduction, extraordinary low-noise performance that turns in very good image quality, can give 47 frames per second at resolution of 1936 x 1216 pixels. The camera can be seen in figure 3.4.

Figure 3.4. *UI-5260CP Rev. 2 camera. [6]*

The camera is placed inside a protecting case in order to protect it from possible hits as it is installed on a heavy machinery vehicle. This case is also in charge of heating and cooling the air inside it, in order to maintain always the same temperature conditions, this way the camera will not stop working in extreme conditions that might occur in Finland. This placement can be see in figure 3.5

Figure 3.5. *UI-5260CP Rev. 2 camera inside protecting case.*

Several tests were performed while trying to center the demountable for further pick up. One of the test had the camera mounted on the top back part of the diver's cabin in the vehicle; the second one, have it on the back bumper of the vehicle. As explained before, this position was selected to have the sensors closer to the demountable and together, this will give the most similar data possible from each of the devices.

Having this new configuration of the camera could avoid mistakes at the time of the hitch detection, as from the top of the demountable sometimes the interior of the demountable was seen and some of the objects inside could cause the software to think the hitch to be inside the demountable instead of on the outside, also, in this position, depending on the angle of the hitch, it could seem a straight rod and not be detected as the curve might not be seen by the camera.

Some of the characteristics of the device are shown in table 3.3.

Table 3.3. *Technical details UI-5260CP Rev. 2 camera (IDS). [6]*

Interface	GigE
Sensor type	CMOS
Manufacturer	Sony
Frame rate	47 fps
Resolution (h x v)	1936 x 1216
Optical Area	11.340 mm x 7.130 mm
Shutter	Global Shutter
Optical class	1/1.2"
Resolution	2.35 MPix
Pixel size	5.86 µm
Functions	Monochrome or color version
	Perfect color reproduction
	Extraordinarily high dynamic range (72 dB)
	Extremely low-noise (6.2 e^-)
	Standard industrial dimensions: 29 x 29 mm
	Global shutter
	Subsampling (horizontal and vertical)
	Long exposure up to 30 seconds
	Power-over-Ethernet
	Integrated 120 MB image memory
Applications	Machine vision applications
	Low-light applications
	ITS (Intelligent Transport Systems)
	Visualization and analysis
	Quality assurance

3.2 Design alternatives

The method selected during this project created the interconnection of the infrared light emitter, 3D sensor, video camera and computer using C++ and ROS, recreating in a 3D space inside Rviz program the objects behind a vehicle.

The steps taken for this procedure started by connecting the 3D sensor and camera to the computer. This computer, when starting the software developed by the Automation and Hydraulic laboratory from TUT, will start receiving data from the 3D sensor once it has power and starts working. This sensor will receive the infrared light when reflected from the objects in front of it, when receiving the light, will calculate the distance and the position of 1024 different measured points, obtained from a 16 x 64 points grid.

Once the grid of points and distances is created, the device will send the information in real time to the computer in form of a point cloud, this point cloud will be visible in the 3D space created in Rviz, having also the position of the vehicle and each of the devices.

At the same time as the above process is being done, the video camera will send real time recording of the scene behind the vehicle, that data will be received by the computer and it will calculate frame by frame the position of pedestrians in the image.

Once the point cloud and the position of the pedestrians are calculated, the software will create, on top of the point cloud, an arrow indicating the normal of the demountable in order for the driver or vehicle to understand how it should turn, in order to get a perfect parallel to the vehicle and place it in the right position.

As explained before, obstacle warning, pedestrian detection and warning, warning signs detection and company logo detection is performed during the project.

Obstacle detection is divided in two parts, object detection and pedestrian detection. In the object detection, the position and shape of the detected objects is obtained from the point cloud data. If some parts of the point cloud fulfill some characteristics, they are considered obstacles. That information is sent to the C++ code, in order to position those obstacles in the 2D image and compare their position with respect of the demountable, if their position interferes with the vehicle path to pick up the cargo, they will be marked as red for the driver to notice and stop the vehicle.

Pedestrian detection follows a similar approach with some changes, for example, extra distance margin is given because they are moving obstacles, this way the driver will have more time to notice them, also, a warning sign is given to the driver if a person is detected in the image independently of their position. For pedestrian detection in the image, OpenCV is used as it has implemented a pre-trained HOG + linear SVM prepared for people detection. Those detections are compared with the position of the detected obstacles and if any those match, the obstacle is marked as pedestrian and treated accordingly. Also, all the points in the pointcloud are checked if a pedestrian is detected in the image, if any of those points happen to be inside the pedestrian region and its position fulfill the conditions for the warning to happen, it will alert the driver.

Also a warning sign detector is implemented using Matlab, this detector will be based in training a R-CNN network using several training images with the corresponding sign, once the network is properly trained, it could be used for real time detection of such signs in the final program, in order to handle with extra care dangerous materials.

Another functionality is to differentiate the company logos that can appear in the demountable, this will be useful in case several companies have demountables inside, for example, the same warehouse; with the vehicle capable of knowing which logo is which, it will be able to understand which cargo should it load or process, leaving the rest untouched.

As extra safety measure, a distance warning is implemented in the code, if any point inside the pointcloud, detected as obstacle or not, is closer than 2 meters to the vehicle, a stop warning sign is shown to the driver to stop the vehicle before a possible collision.

3.2.1 Programming languages

The selected Programming languages for the projects were C++ and Matlab. C++ to create the main program, connecting the computer with the sensors and perform the transformations, the pedestrian detection and warning, obstacle warning and proximity warning. This part of the project will be programmed using Qt Creator as the development environment.

This selection was done as ROS environment is meant to be used with C++ or Python, this way the development of the whole project became more easy to perform. The developers were more comfortable with C++, because they were already trained in its use.

The second language selected for the project is Matlab, with this language, the object recognition part was built, also the R-CNN detector for signs and company logos. Again, this programming language was selected because of the previous experience of the developers using it.

The final program interconnects both languages, C++ and Matlab, for obtaining the desired results, having Matlab code generated and compiled into C++ code for merging. ROS environment is also used to perform transformations to the data obtained by the devices.

3.2.2 Libraries selection

The libraries selected go from the ones that are related with the sensors to the ones that will output the data in real time in Rviz window going through all the libraries needed for each step inside the code. This will make the program able to manage all the data and the appropriate calculations performed to it.

Those libraries are mainly used inside the C++ code as it does the bigger part of the work. ROS environment will be the main tool during the project and all the data will go through it before and after processing it.

3.3 Libraries

In this section the libraries selected for the project in Matlab and C++ are shown for a better understanding of how the overall program works and its connections with other parts of the software.

3.3.1 Matlab libraries

Several Matlab toolboxes are used during the implementation of the project, those toolboxes are:

- Statistics and Machine learning Toolbox, this toolbox provides functions to describe, analyze, model data, feature selection, stepwise regression, principal component analysis and regularization. This toolbox offers supervised and unsupervised machine learning algorithms, as SVMs or K-nearest neighbor used in this project. [36]
- Image processing Toolbox, this toolbox provides a set of reference-standard algorithms used in image processing and analysis. [37]
- Computer Vision System Toolbox, provides algorithms prepared to simulate computer vision and video processing as feature detection, extraction and matching. This toolbox is used for object detection and recognition during the project. [38]
- MATLAB Coder, generates readable and portable C++ code from Matlab. [39]

3.3.2 C++ libraries

The main libraries used inside the C++ part of the project are the following:

- ROS libraries to subscribe to the data needed and publish the output into topics are needed during the whole loop, this way the data recorded in real time is processed and the results are shown to the driver.
- Libraries to perform coordinate transforms "tf" are needed in the program, ir order to obtain the right positions of the data in 3D with respect to the devices and vehicle.
- Visualization_msgs library, this is used for publishing and presenting the object and other data in rviz environment.
- Point Cloud Library (PCL), it is used for understanding and detecting objects from point cloud data and store it in the right way for the rest of the program to be able to use it in further calculations.
- OpenCV, is used for image processing inside the program, in this part of the project, is used mainly to perform the pedestrian detection, also to mark the obstacles and pedestrians as problematic or not.
- Pinhole camera model libraries are also used as, some of the tasks performed, need the transformation from 3D data to 2D in order to use it in the image.
- Also other libraries regarding the sensors and how the information calculated during the process is shown to the user are included at the beginning of the program. As well as some libraries to handle files and mathematic operations.
- During the program tests, it is important for the developers to know how much time the whole algorithm takes from start to end, for that reason, time libraries are added to measure processes performance.

- Libraries regarding communication through CAN buses are included. This communication is the one used between the 3D sensor and the computer and could also be used in future works when sending information to the vehicle to move itself.

3.4 ROS environment and Rviz

ROS is the main tool in this project, all the test and calculations are made through it, using as interface for the user Rviz program. In Rviz window, it is shown in 3D space the position of the vehicle and devices attached to it, those positions are calculated using transforms, those transforms will make them placed in the right position, in order to make the appropriate calculations when performing tests.

It can be seen in real time inside the 3D space, the point cloud generated by the sensors with respect to the vehicle, in order to locate the demountable or obstacles that could be in the way to it. The camera image is also shown in real time and, on top of the image, the point cloud demountable and hitch position detections are overlayed. This is useful during testing to understand how to tune the software or hardware if needed.

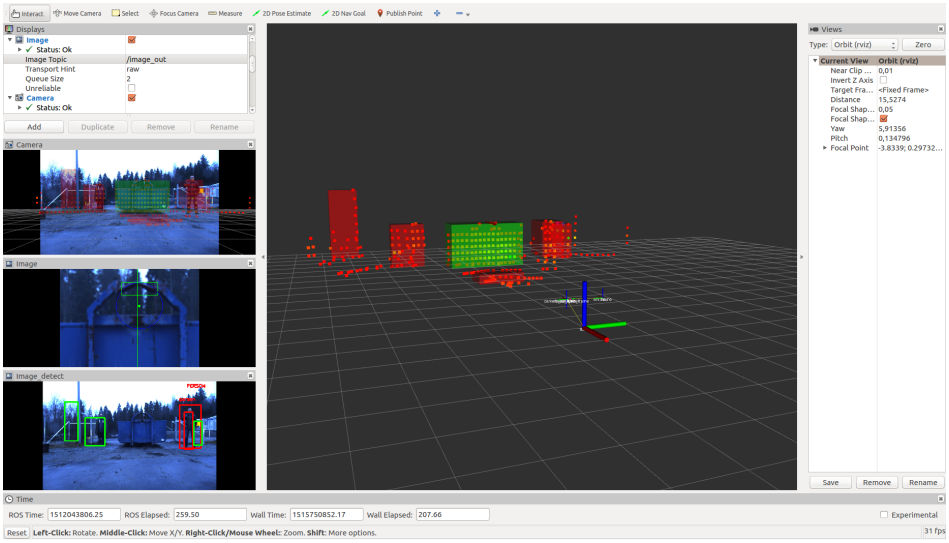

Figure 3.6. *Example of test data visualization in Rviz.*

In the image 3.6, as explained in 2.2, the reader can observe on the left part some configuration options regarding the visibility of certain aspects of the data, also options for possible displays during the use of Rviz.

Under it, the window called "camera", is in charge of showing the real time video, as well as the poincloud positioned on top of it as an extra test for the developers to check if the data after transform is correctly placed to continue the process. This test is very useful to understand how to change some parameters in the code if needed. This characteristic has been very helpful in the process, in order to detect obstacles and marking them as safe for the vehicle, to approach the demountable or not.

Under that window is located the "image" window, this window will show the ROI of the image, in this case the demountable part of the image, from the recorded image that is being read from a .bag file stored in the memory of the computer or an external disk; most usually computer during this project due to the size of the files.

The last window, appearing on the left, is "image_detect", here the obstacle and pedestrian detection will be shown to the user in real time with their corresponding warnings; red rectangles, stop and pedestrian.

In the middle part of the Rviz window, the 3D space is created by the program, inside this space all our nodes, vehicle, 3D sensor and camera, will be represented as axis coordinates, the point cloud recorded by the 3D sensor will also be shown here. This point cloud is the recreation from the objects located behind the vehicle. In the image the big green rectangle in the center would be the demountable to which the vehicle was approaching.

The last part of the window, in the right part, information about the view point, where and how the 3D view is shown is presented to the user.

As the recorded video is saved in raw format, the memory size required for each test .bag file can be several gigabytes, storing in the same file all the information from the camera and the 3D sensor.

3.5 Coordinates transform

The coordinates transforms of the devices to position them properly in the 3D space created in Rviz, is performed using ROS environment. ROS gives some tools to do such transforms in a simple way. The code will receive the package in which such transformation can be found, the type of transformation and the frame that should be transformed, with the change in the coordinates as X, Y, Z position and the change in the angle of each axis. The name of the new output is also given, this will be the name that will appear in the 3D space created in Rviz when placing the device in it and finally, the output place for the transformation. An example of how the transformation looks inside the .launch file of the project is shown in the following piece of code.

```
1 <node pkg="tf" type="static_transform_publisher" name="camera_tf" args
    ="0.0 0.0 0.0 1.57079632679 0.0 1.57079632679 /back_frame /camera
    100" output="screen" />
```

3.6 ROS environment start

The first step when trying to analyze or record new data, is to start ROS environment. As ROS is build on top of Linux, a computer with Linux software is needed. In the case of the project, Linux Ubuntu 16.04 distribution is used.

To record data, ROS has an already implemented command that stores the data inside .bag files, such files will be used by ROS to read and send the information to the developed program, taking into account the time of the recording to avoid errors when running.

To see the results of the old data in Rviz, some steps are needed. First of all, the ROS environment should be started; in a new terminal is needed to use the command "roscore", this command will start a collection of nodes and programs required for ROS to run properly as a Master, a Parameter Server and a rosout logging node.

Once ROS is started, one command will specify to use the simulation times instead of the real time, in order to avoid problems when computing the position of the point in the point cloud or placing the markers on top of the image. For this "rosparam" command is used.

```
1 rosparam set use_sim_time True
```

This command is the one stated in ROS for getting and setting parameters that appear inside the Parameter Server started previously.

Now Rviz program can start, this program will show, after loading the desired configuration of the window, all the data that was recorded and computed in the program in "real" (simulation) time. To start Rviz, the following ROS command is used.

```
1 rosrun rviz rviz
```

This command allow the user to use a package without giving the full path to ROS. Basically, the first parameter will tell ROS in which package it should look for the program to run, the second parameter tells the program name. Rviz window after selecting the desired configuration can be seen in figure 3.6.

After Rviz is running, the user should initialize ROS environment in order for the packages used to be found. To do so, the first step is to set the ROS workspace. It is done using the following command.

```
1 source devel/setup.bash
```

Basically, this command specify as workspace the one located in the path given (setup.bash inside devel folder).

When the workspace is initialized, "roslaunch" command will be used in order to get the desired transform in the position of the devices inside Rviz space. This command will launch the desired program, stating all the needed connections between nodes subscribed or publishing and topics, as well as the transforms performed on them to obtain the appropriate output in Rviz.

```
1 roslaunch launchFolder vehicle_06_11_17.launch --clock /tf:=/
    tf_old
```

If such command is not used, the old configuration of coordinate transforms would be used giving us the wrong positions of the devices, as they were moved to the back bumper of the vehicle as second test. Now the new coordinates, calculated after the transform of the original ones, will appear in Rviz 3D space whenever data is played on it.

At this time, the user can check how the program is performing with the data recorded previously. One should also take into account that, if there are some changes in the code or in the launch files, the program should be compiled again using the command "catkin_make" in the root folder of the workspace once the workspace is set on top of ROS using the "source" command line.

3.7 ROS launch codes

The way for Rviz to understand which nodes will be connected to it, also, which will be subscribers or publishers to some topics, is the .launch file. This file will, basically, tell ROS all the nodes that should be launch in a program and will set all the parameters needed.

The first step in the code, is to set the parameters needed by the program, for example, the threshold for Canny edge detection. The parameters initiated are the following:

```
1  <param name="RawTopicIFM" value="/ifm_o3mxxx_raw" />
2  <param name="RawTopicCamera" value="/camera/image_rect_color" />
3  <param name="analyze_frame" value = "camera_analysis_frame" />
4  <param name="ifm_analyze_frame" value="ifm_frame" />
5  <param name="canny_thresh" value="30"/>
6  <param name="dist_limit" value="0.07"/>
7  <param name="normal_limit" value="0.6"/>
8  <param name="arc_template_path" value="/home/VEHICLE_BAGS/06_11_17/
       template_simple4.jpeg"/>
9  <param name="Flip" value="True"/>
10 <param name="template_width" value="0.35"/>
```

Those parameters specify the data recorded from the 3D sensor, the data recorded from the video camera, the frame that processes the information to get position of the demountable and the hitch, threshold used for Canny edge detection, the distance limit between point for them to be considered from the same object and normal limit, used also to know which points belong to the same object, the path to find the hitch template used for object detection, "flip" parameter used in case the recorded image needs to be flipped; finally, the width of the template used.

Now, all the transformations used to position the devices in the proper position, inside the 3D space created in Rviz, are written, they will change the original position and angles from the devices, as they were moved during the different tests performed in the project.

```
1 <node pkg="tf" type="static_transform_publisher" name="camera_tf" args
    ="0.0 0.0 0.0 1.57079632679 0.0 1.57079632679 /back_frame /camera
    100" output="screen" />
2 <node pkg="tf" type="static_transform_publisher" name="analysis_tf"
    args="0.0 0.0 0.0 0 0.0 3.14159265359 /back_frame /
    camera_analysis_frame 100" output="screen" />
3 <node pkg="tf" type="static_transform_publisher" name="back_frame_tf"
    args="0.0 0 0.5 3.14159265359 0 0 /vehicle /back_frame 100" output
    ="screen" />
4 <node name="ifm_frame_tf" pkg="tf" type="static_transform_publisher"
    args="0 0.2 0.0 0 0 0 /back_frame /ifm_frame 100"/>
5 <node name="ifm_tf" pkg="tf" type="static_transform_publisher" args="0
    0 0 0 -0.05 0 /ifm_frame /ifm 100"/>
6 <node ns="/camera" name="image_proc" pkg="image_proc" type="image_proc"
     output="screen">
```

The information in those lines will tell ROS the package in which the transformation is ("tf"), the type of the transformation, the name of the output, the actual transformation in X, Y, Z and normal angles format, the frame that receives such transform (back_frame) and how many times the data is published (every 100ms), finally, where the output will be published.

With this ready, the program is prepared to go through the C++ code, at the same time it shows in real time in Rviz the 3D point cloud captured by the 3D sensor, also, the images recorded by the video camera with all the computations performed in the code.

3.8 Data recording

During every test, the team performed several tries to get all the data needed for the processing stage, for example, driving the vehicle backwards from several angles, turning the vehicle and changing the demountable to check whether or not the software was able to perform properly all the tasks.

The data recorded comes from both, the 3D sensor and the video camera, to be able to check and compare positions of the object in each set of data overlapping both data sets in the same image.

As explained in section 2.1, the TOF device will obtain the position in 3D space of each point in the grid. The recorded data is transfered to the computer via Ethernet cable. The video-camera, basically records real time video and transfers the data to the computer, it will be transferred via Ethernet cable.

Once the data arrives to the computer, the software encapsulates it in .bag files, those files are the ones needed to read and use the information in the ROS environment. The program stores the data using the command shown in after this paragraph.

```
1        rosbag record /ifm_o3mxxx_raw /camera/image_raw /camera/
            camera_info /tf          /tf_static -o vehicle
```

- rosbag, this is the actual command that provides functionality with ROS bags (file formats to store ROS messages).
- record, this part gives the instruction that data will be recorded into a bag file with the specified contents following.
- /ifm_o3mxxx_raw, this is one of the contents that will be stored inside the .bag file, in this case this will be the data recorder by the 3D sensor.
- /camera/image_raw, this is the next content in the .bag file, it is the raw image recorded by the video camera installed.
- /camera/camera_info, this is the next information that will be stored in the .bag file, it represent the camera parameters as for example calibration parameters.
- /tf/tf_static, finally, this is the last information that will be stored inside the desired .bag file with the rest of data from the camera and 3D sensor, it will determine the static transforms made in the recorded data to change its position into the appropriate space for processing it.
- -o, this command tells ROS to start the name of file with a name given by the user.
- vehicle, this is the name given by the user to start the actual name of the file.

After that command is typed, ROS will be able to store the information in a new .bag file, that file will be created every time the recording starts, such .bag file will be stored in the "BAGS" folder, inside the project workspace. The new file created will have a given name followed by the time stamp of the date and time the recording started.

Once the data is stored, it can be used again, simulating real time cases using ROS and tuning the parameters if needed, also is possible to watch the results in Rviz, as discussed in section 3.4.

3.9 Data preprocessing

For the program to work as expected, some preprocessing is needed before and while using the data inside the C++ code, for example, transform the coordinate space, as explained in section 3.5, in which the devices were placed in order to have the good positioning of the data.

Also, in the initial data, is important to remove the data recorded representing the ground, as this will save time and computational power when running the program. During the program, a region of interest inside the total image is cut in order to perform some extra operations on it, this will again save time and computations to the computer, as the area where it should perform such calculations is much smaller.

Also, to perform the pedestrian detection, the image from the data recorded is duplicated and reduced to 0.3 its original pixel size, this will make that all the calculations needed to be performed for the pedestrian detection, to be less in number, making at the same time the process much faster to achieve real time detection in the image. As the reduction rate is given by the developers, all the calculations performed with this data will not affect other operations, as it will be programmed accordingly to avoid mistakes.

The poincloud and detected demountable, will be placed in a wrong position if the coordinate transform is not the appropriate one. This will make the transforms to not have the appropriate calibration of the devices, this way, the computations done by the program will give wrong results.

In the same way, in figure 3.7, it is shown how the pointcloud and detected objects are properly placed on top of the image, as this transform is the relevant one. Now the devices are placed in the right position with respect to the vehicle.

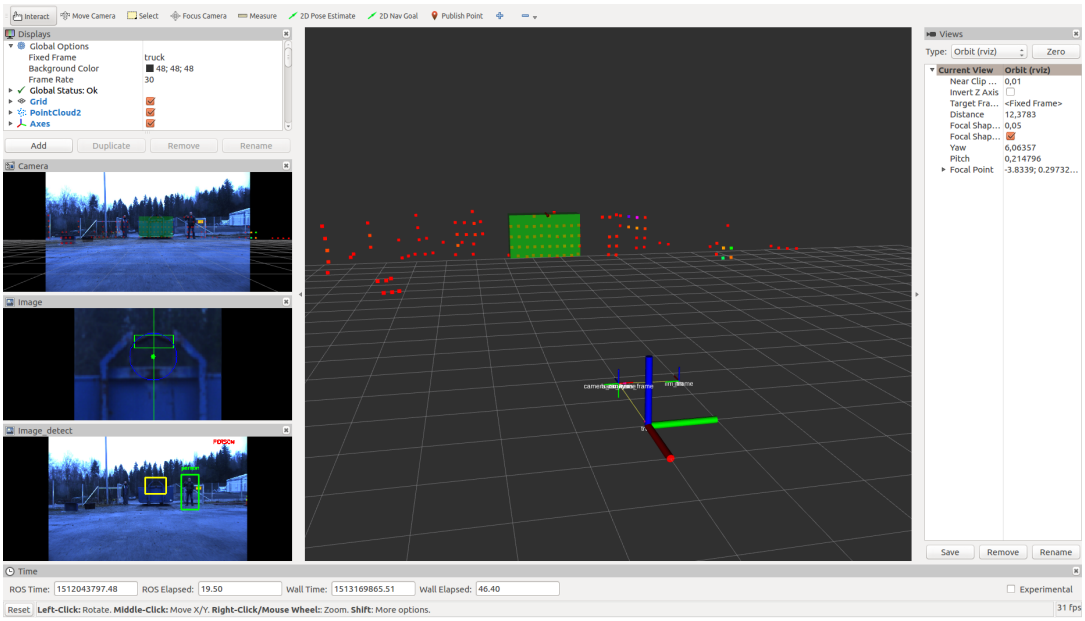

Figure 3.7. *Rviz data shown with appropriate launch file.*

3.10 Obstacle detection using IFM data

Obstacle detection is performed first in the obstacle detection code, once the obstacles are detected, their position will be checked in order to understand, whether or not, they could be a problem when approaching the demountable. IFM data is used for this task.

First of all, the obstacle detection code will detect the corner points of any part of the pointcloud having points resembling an obstacle, then it will classify such obstales as only obstacles or as demountables. Afterwards, such markers will be shown in Rviz as red boxes as shown in figure 3.8.

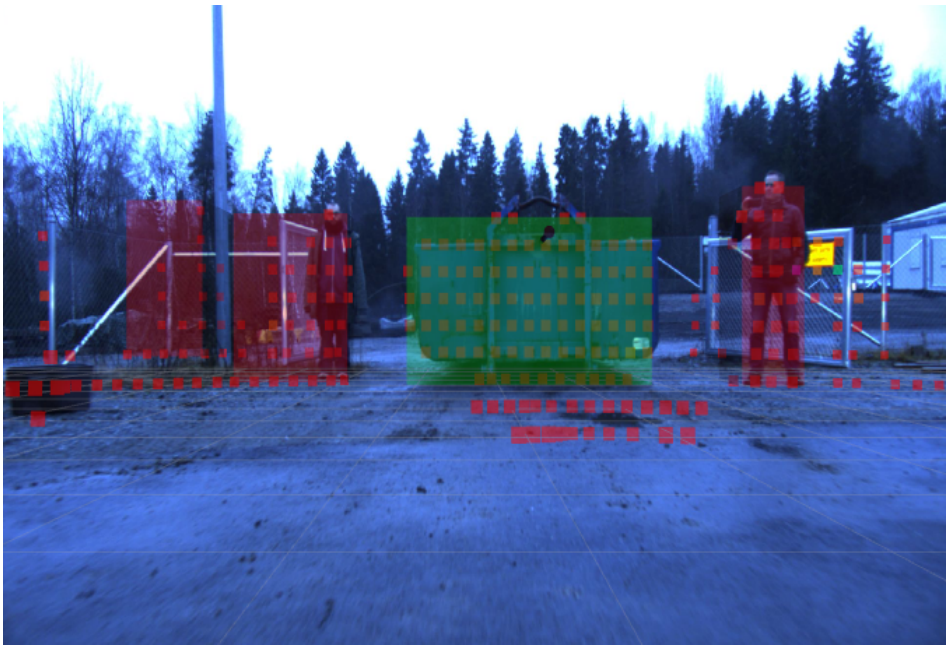

Figure 3.8. *Detected obstacles from 3D data overlayed in 2D image of recorded data.*

Those boxes are placed as rectangles in the 2D image, after their position is checked in the pointcloud, to know if they will affect or not to the vehicle path as shown in figure 3.9.

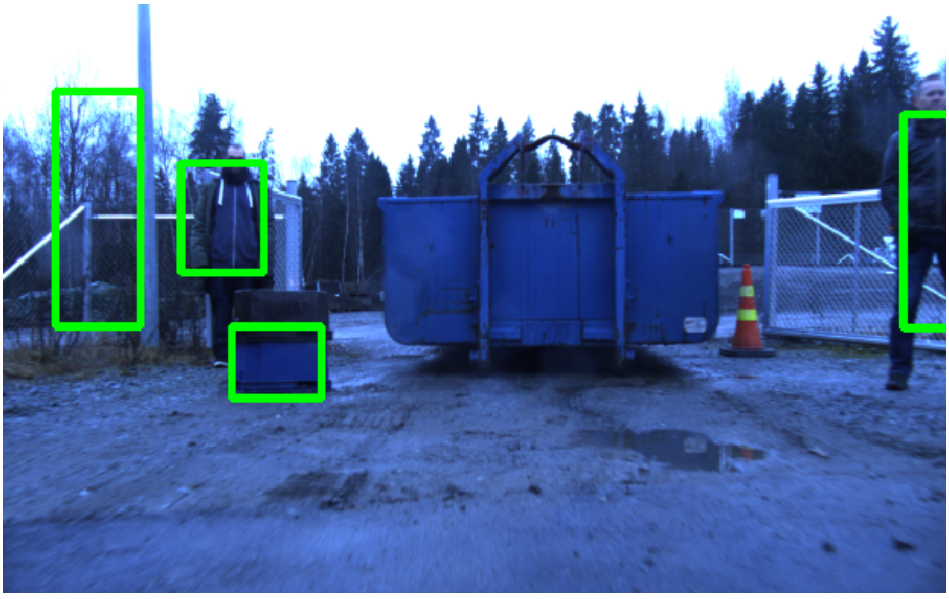

Figure 3.9. *Obstacle marking in image using recorded data.*

The method to check if the obstacle is on the vehicle path is the following:

- Object position higher than ground level, to avoid possible close ground points in the pointcloud as detected obstacle.
- Object position closer to the vehicle than the demountable, as they can only interfere with the vehicle if the obstacle is placed in front of the demountable.

- Object closer than 0.5 meters to the side of the demountable, As it can move a bit to the sides when being picked up, a safety margin is given to avoid hitting the obstacles. This distance is calculates using the corner points of the obstacles (left corner point if obstacle is positioned on the right of the demountable and right corner point if the obstacle is positioned at the left side of the demountable). Such corner point position is calculated using the middle point position of the demountable, as the width of it is known. This safety margin can be seen in 3.10.

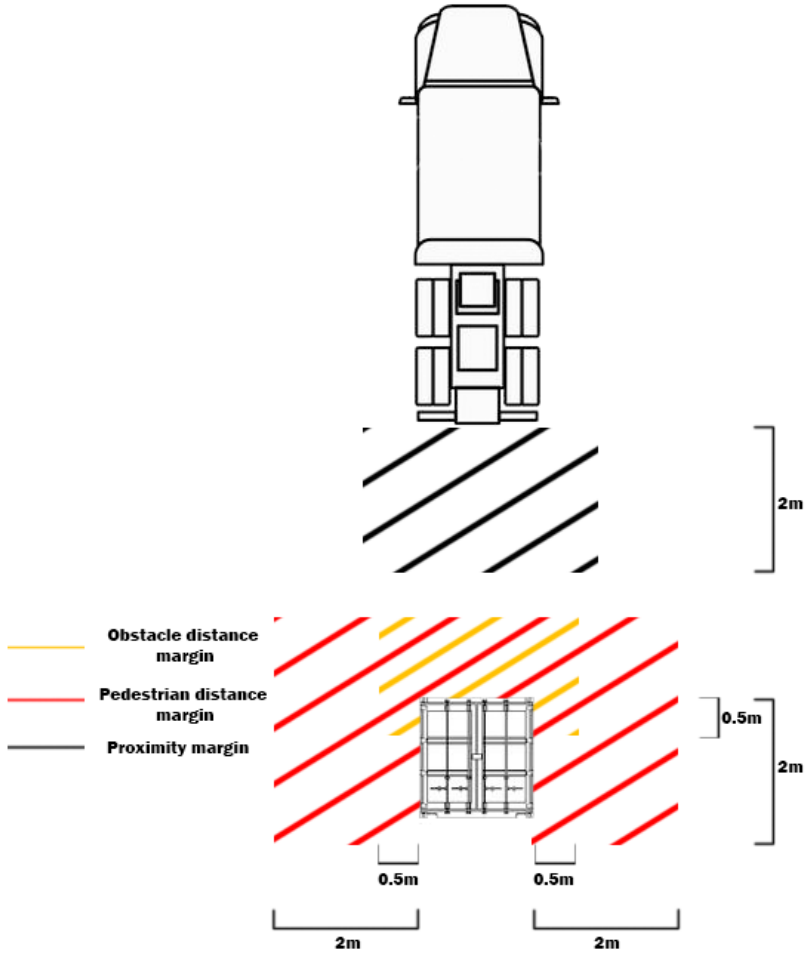

Figure 3.10. *Safety margins given to vehicle, pedestrians and obstacles.*

If the obstacle does not fulfill those safety measures, it is marked with a red box to indicate the driver to stop, otherwise, is marked green meaning no danger to the process. This can be seen in figure 3.11.

A variable to change the allowed side distance to the demountable is implemented, as default, it is set to 0.5 meters for objects and 2 meters for pedestrians but in figure 3.11, obstacle margin distance is set to 2 meters without taking into account if it is an obstacle or a pedestrian. It can be seen that obstacles closer than that distance are marked as red, obstacles further away are marked as green.

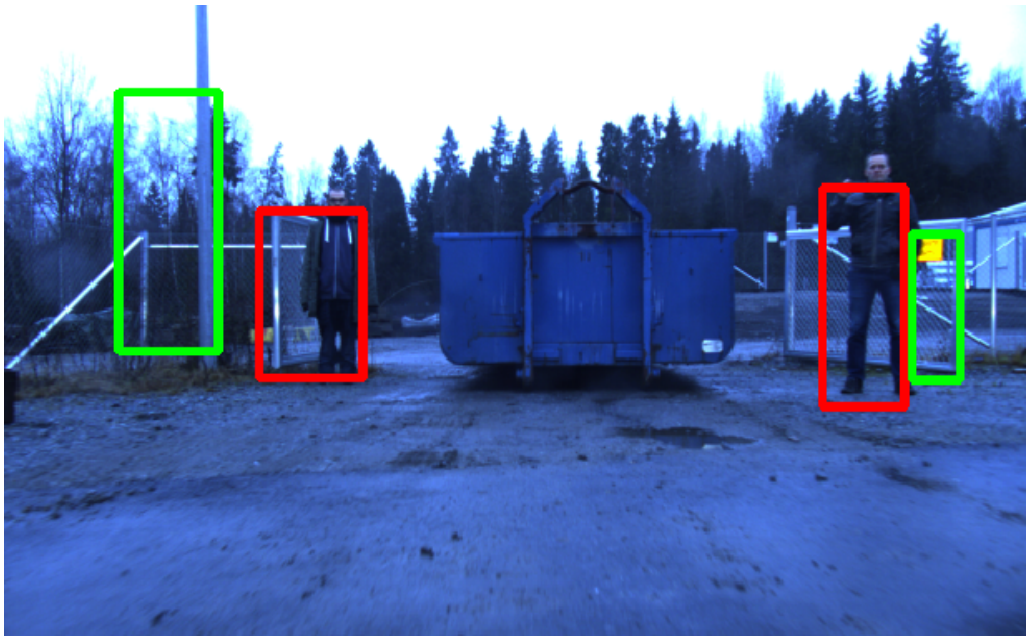

Figure 3.11. *Obstacle warning using recorded data (2 meters distance margin given with no demountable front distance check).*

3.11 Pattern recognition and warning

The main goal of the project is to perform pattern recognition in the recorded data, in order to find the demountable, pedestrians and warning signs, while detecting security issues that might affect the loading of the cargo into the vehicle.

To do such task, pedestrian detection will be implemented in C++, while warning sign detection will be implemented in Matlab. Obstacle detection code will receive the recorded preprocessed data before the object recognition part, then it will perform the actual detection of the obstacles and demountable. The selected ROI for the pedestrian detection is the whole image, it will be checked accordingly to find the correct position of all pedestrians.

Once the position of all the obstacles in the image is set, those positions will be at the same time compared with the detected pedestrians in the image, if the position of the obstacle is more or less similar, or it is inside the ROI of the pedestrian, it will be marked as pedestrian; then, the process to mark it as dangerous or not for the vehicle movement will change.

While the above process is performed, if a pedestrian is detected, all the position of the points in the pointcloud are checked, in order to have an extra safety measure in case the pedestrian is at a dangerous position when moving the vehicle.

To detect the pedestrians, the program uses a pre-trained neural network ready to perform the pedestrian detection in the desired image or images, in this case, our real time video, this detection needs certain parameters to tune the most ideal detection possible, trying to avoid false positives and false negative detections. Once the tuning is ready, the next step inside the pattern recognition part can start.

While the pedestrian detection is being done, the program could use the ROI of the demountable, in order to check for possible warning signs in it as flammable or explosive signs for example; this will reduce the computer power needed to perform the detection of such signs in the whole image. To perform such task, a deep neural network is trained with the desired patterns to detect, in our case flammable, explosive and biohazard signs. When it is trained, it will be ready to perform the desired recognition in any new image used as input.

The same procedure as the one just explained could be used to detect the logo of the company in the demountable, in order for the vehicle to know which demountable it should or should not load as cargo. To perform this task, the same method as the one used for the warning signs detection is used.

3.12 Pedestrian detection

One of the parts of the project is to determine whether or not pedestrians are placed behind the vehicle and when their position could be a problem when the vehicle is reversing to pick the demountable.

3.12.1 Movement-based pedestrian tracking

The first test to perform pedestrian detection during this project was using Matlab code, in order to perform several object tracking based on movement, this method would obtain real time video data recorded by the video camera placed on the vehicle. This method would have several parts to obtain the resulting tracking as: [40]

- Background substraction, based on Gaussian mixture models, this is a key point as without having the program understanding the background the detection of the pedestrians could not be performed.
- Noise deletion, this will help to obtain the cleanest possible detection during the process, this noise deletion is performed on top of the foreground mask of the image.
- Blob analysis, this part of the program would detect connections between image zones that could represent the same object moving.
- Kalman filter, it is responsible to understand the path of the moving objects, to mark them as detected in the output image. This process tracks the object only if it is marked during several consecutive frames, in order to avoid false positives, in the same way, if the object dissapear during several consecutive frames, it gets unmarked.

How this method looked during the first stages of the project is shown in figure 3.12.

Figure 3.12. *Movement-based pedestrian detection output. (JureKreft)*

And the corresponding movement mask for the frame shown in figure 3.12 can now be seen in figure 3.13.

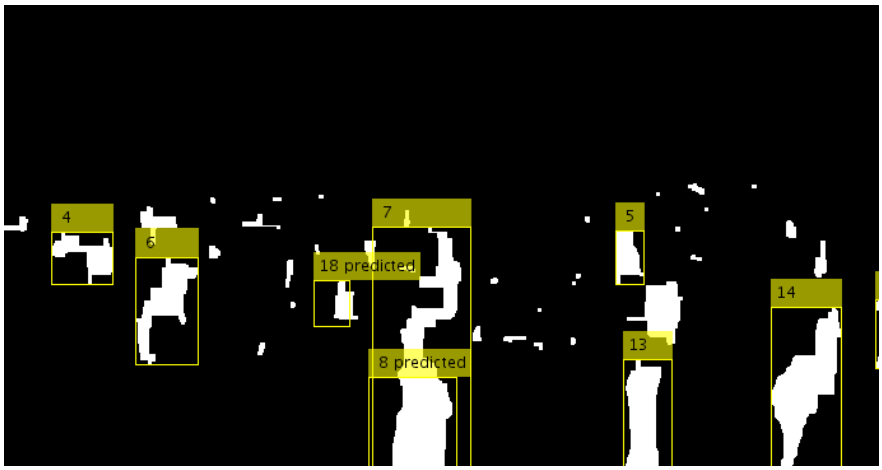

Figure 3.13. *Movement detection mask.*

This method was quickly discarded, as there was no sense into having a motion-based detection while the camera is not static most of the time as it is placed in a vehicle that must move to pick the demountable.

3.12.2 Pedestrian tracking using Aggregate Channel Features (ACF) detector

The second attempt to obtain pedestrian detection using Matlab was performed using the "peopleDetectorACF" feature already implemented in Matlab. This is a pretrained detector, which training allows to detect pedestrian in an image or, in our case, a set of images forming the real time video when the vehicle approaches the demountable to perform the pick up. [41]

The detector, an "acfObjectDetector" inside Matlab, was trained using INRIA person dataset.

During this process, the video is loaded into Matlab's workspace and the frames will be read one by one until the end of the video. Each time a frame is read, the pedestrian detection is performed in the image, looking for patterns similar to the ones given to the detector during training, in this case, the detector training is based on Aggregate Channel Features. Once the detection is perfomed in the whole frame, the output are the bounding boxes where the positive detection appears, marking the region in which the program thinks there is a person and the score given to that region, this score represents numerically on a scale from 0 to 100 how close is the detected pattern to the one in the trained detector, the higher the value, the closer it is to the ideal detection. The output of such detection can be seen in figure 3.14.

Figure 3.14. *Pedestrian detection with bounding box and score in Matlab.*

This method required for some tuning to achieve real time detection in the sequence of images but, even though such goal was fulfilled, this method was discarded as some other method with, basically, the same results was obtained directly on C++ using OpenCV features. Using this new approach, saved one extra step for the project as the conversion from Matlab code to C++ code was not needed.

3.12.3 Pedestrian tracking and warning using OpenCV

The next approach performed during the project to detect the obstacle represented by the pedestrian, is similar to the one used for detecting any obstacle, the only difference is that, as pedestrians are moving obstacles, extra distance margin to the demountable is given, as they could move to a problematic position is short time. This margin was explained in section 3.10 and can be seen in figure 3.10

Basically, this detection is performed after object detection as described in section 3.11. If the detected obstacles does not fulfill the safety measures, should be marked as red, but, an extra step is performed before marking them, the pedestrian recognition in the image.

Pedestrian recognition

For the pedestrian recognition, OpenCV libraries are used inside the C++ code, with those libraries, it can be used a pre-trained HOG + linear SVM network, capable of detecting the pedestrian in single frames of the video. It can detect both, single pedestrian or several ones inside the same image.

To have real time pedestrian detection, this step will be performed for each frame for the video in the loop that sends all the information recorded, 3D sensor and video camera, for the processing of it. This detection process can create a bottleneck for the computer memory if the parameters used for such detection are not tuned properly; that will stop the program for some time avoiding the real time process to continue.

Once everything is set to store the needed data, the new HOG descriptor is created, this HOG descriptor will be said to use the pre-trained network for pedestrian detection inside OpenCV.

The first, and one of the most important parts to save computer power and memory, is to resize the image in which the detection will be performed, this will almost not affect to the detection, if the selected size is properly chosen but will reduce enormously the number of operations performed on the image. In this project, images are resized to 0.3 the original size of the image in pixels in both, X and Y axis.

Now that the descriptor and the vectors are ready, the detection process can start. Basically, the program is given a command that will perform the detection using several parameters on the desired image, the line looks as follows:

```
1          hog.detectMultiScale(image, found, 0.57, Size(4,4),
             ↪ Size(8,8), 1.1, 2);
```

This line parameters include: [42]

- Image in which the detection will be performed.
- The name of the output vector of rectangles indicating the detections.
- Hit threshold given for detections, every detection is given a score indicating how close to the comparison template is, depending on light conditions, blur of the image due to movement or the shape of different objects this number can affect to select some object the image as a pedestrian (false positive), or to not be able to recognize a pedestrian as one (false negative). Tuning this parameter, the developer can check and select the most relevant approximation to the truth captured by the camera.

- Window stride, this parameter indicates the step size taken by the window in x and y dimensions, this parameter is very important, because a big jump can reduce computer power significantly but might also avoid the detection of some of the pedestrians in the image.
- Padding, this parameter indicates the number of pixels in which the sliding window will be padded in both axis during the process. This parameter tuned properly can increase the detection accuracy.
- Scale, again, this parameter is one of the most important to take into account when balancing computer power and detection accuracy. This parameter indicates how much the image is scaled each time to be compared with the HOG pedestrian template. If a big step in the scale is chosen it might lead to false negatives, but if the step is too small it will increase the time and memory needed for the process significantly.
- Group threshold, this parameter indicates, whether or not, perform grouping in the detections inside the image. As there is a sliding window, one single pedestrian can have several detection rectangles (figure 3.15), this parameter will take into account the position of those rectangles and if they collide, they will be grouped as one single detection (figure 3.16).

***Figure 3.15**. Pedestrian detection without grouping.*

***Figure 3.16**. Pedestrian detection with grouping.*

As the detection tends to increase the size of the pedestrian detected rectangles, after the detection process is finished, all rectangles are shrinked a bit to adjust better to reality. Once this step is done, the pedestrians are marked in the image as in figure 3.17.

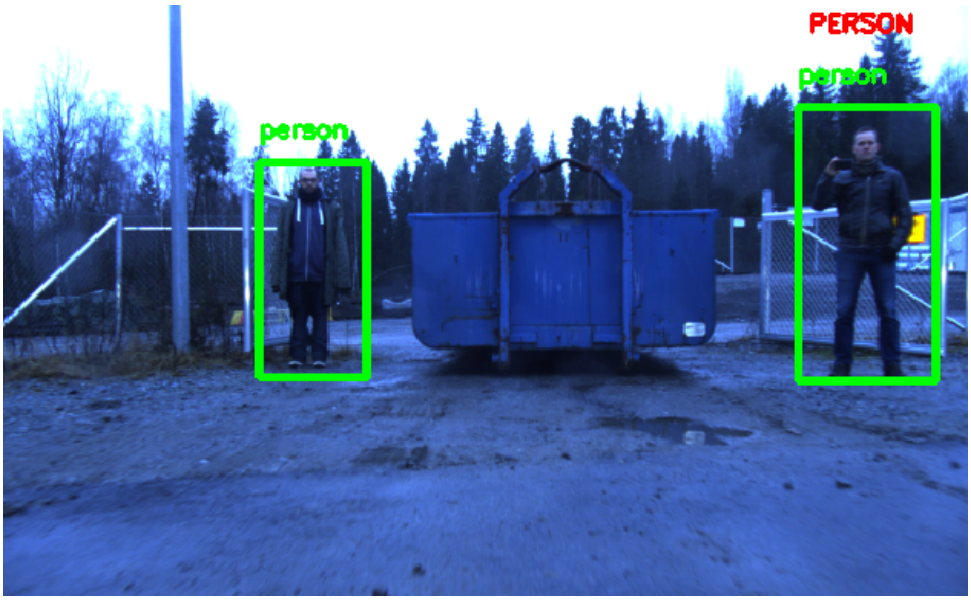

Figure 3.17. *Several pedestrians marked in a single frame.*

Once the detection is performed, there are two extra processes to complete. In the first one, every time one or several pedestrians are detected in the image will show a red "PERSON" sign on the top right of the image, this way the driver understands that pedestrians are walking by and that, he or she, might need to be extra careful when moving the vehicle. This sign appears despite the position of the pedestrian as seen in 3.17.

The second process is to check if the position of the pedestrian might or not be a problem when the vehicle moves, this is the process explained above about the safety margins to the demountable.

Pedestrian warning

Each obstacle position detected by the obstacle detection code will be compared with the position of the detected pedestrian rectangle. Basically, this process will transform the obstacle box coordinates into another rectangle in the 2D image using the Pinhole camera model, those rectangles positions will be compared with the position of all the pedestrian detected rectangles, if the coordinates of both rectangles are similar inside the image, giving some margin, as the detections are not ideal, both, the obstacle rectangle and the detected pedestrian rectangle, will become red as a warning indicator for the driver. 3.19

In figure 3.18, it can be seen how, even though the obstacle marker is inside the pedestrian marker, it is not marked as warning, this is because the detected obstacle was not the pedestrian but the fence behind it, the program checks the position of both, the pedestrian and the obstacle and determines that, as it is a different obstacle it does not follow the pedestrian safety measures.

At the same time, in figure 3.19, it can be seen how, when the obstacle detected is the pedestrian and the comparison of positions between the pedestrian and obstacle does not fulfill the safety measures, both markers become red to indicate the driver to stop.

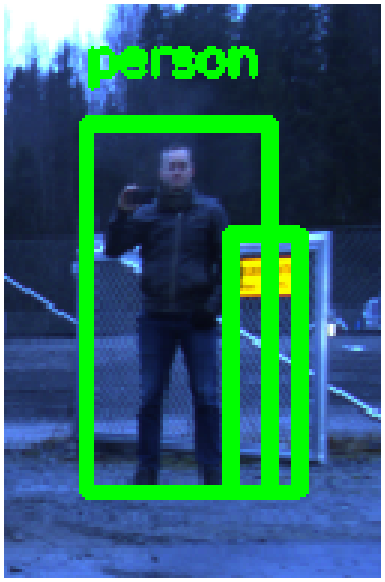

Figure 3.18. *(a) Pedestrian detection without warning.*

Figure 3.19. *(b) Pedestrian detection with warning.*

In figure 3.20, both cases appear at the same time, the pedestrian obstacle is marked as red, making the detected pedestrian marked become also red, while the fence is marked as green as its position does not make it be considered the pedestrian.

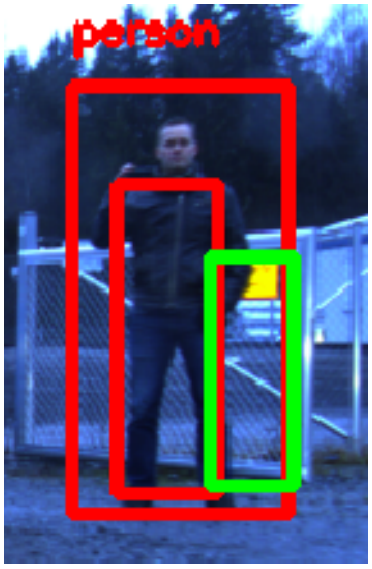

Figure 3.20. *Several obstacles detected in the same space but only one is detected as pedestrian obstacle.*

This process might not work always as, due to the resolution of the sensor, the points inside the pointcloud that would represent a pedestrian are not always considered an obstacle.

To avoid such problem, a new implementation for the warning sign was developed. With this method, instead to wait for the pedestrian to be recognized as an obstacle, every time a pedestrian is detected in the image, a loop will start; this loop will check the position of every point in the pointcloud with respect to the position of the bounding box of the detected pedestrian in 2D and with the position of the demountable in 3D.

This process will, basically, transform the pointcloud according to the video camera calibration parameters, in order to use this new coordinate system to compare positions with the demountable. Also, converts the pointcloud into 2D coordinates to make comparisons with the pedestrian bounding box inside the 2D image in UV coordinates.

Once the transformations are performed, the process to warn the driver, in case the pedestrian is in a position that might obstruct the approach to the demountable with the vehicle, is similar to the process described before with some minor changes.

The first step will be, again, to avoid points considered ground detections, to save computer power. The next step, will be to check if the point in 2D space in the image is inside the bounding box of the pedestrian; if it is, the distance of the point to the vehicle will be compared with the distance between the demountable and the vehicle, if this distance is smaller than the one to the front part of the demountable plus 2 meters margin, as pedestrians are moving obstacles, the final check before the warning sign will start. This last check, consist in computing the distance of the point to the side of the demountable, if this distance is smaller than 2 meters margin given for pedestrians, the warning sign will finally appear marking the bounding box in red for the driver to notice that such pedestrian might block the way of the vehicle. This is shown in 3.10.

Every time the warning sign appears, the loop checking all the points in the point cloud stops, this saves computing power and time, as there is no need to keep checking extra points when one already gave enough information to know that there might be a problem while approaching the container.

The loop checking all the points is performed for each of the pedestrians detected in the image, assuring this way, full control while approaching as this new method will check in every frame that there is no problem with pedestrian postions while approaching the demountable.

At the same time this process is being performed, the program draws all the checked points in the pointcloud on top of the image, for the developers to understand better how the process behaves during the approach.

Figure 3.21 did not have the 2 meters distance margin from the front of the demountable, to check whether or not the program behaved properly. It can be seen that, even though the person on the right was considered a possible problem when approaching to the demountable, the person on the left did not, because of that reason all the points in the pointcloud were checked and draw in the image.

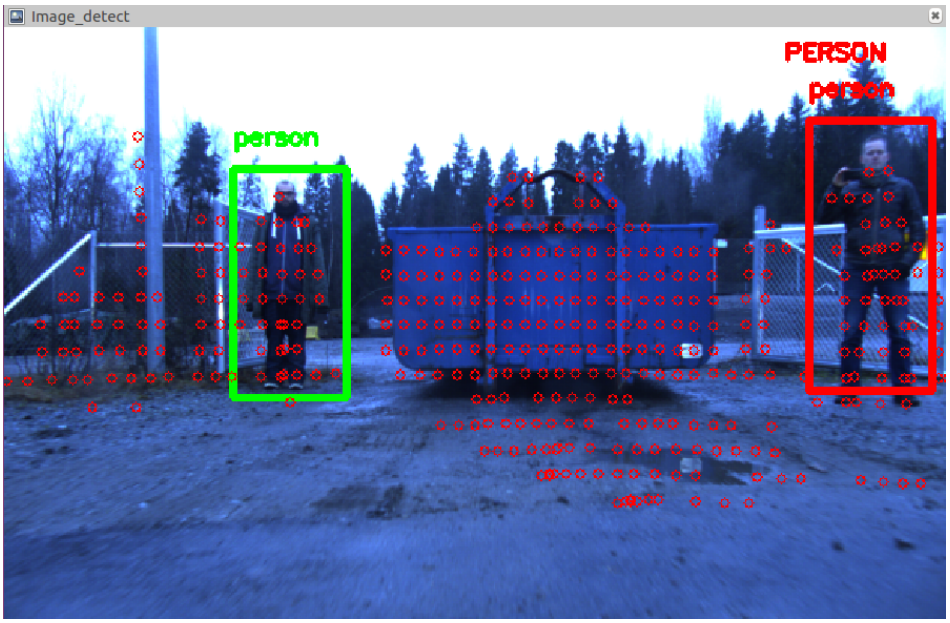

Figure 3.21. *Pedestrian warning without demountable front distance check.*

The final result, with all the security margins applied in the pedestrian warning can be seen in 3.22. In that figure, it can be seen that only 6 points were drawn as the warning had enough information to be set already.

Figure 3.22. *Pedestrian warning with all safety measures.*

To try to get the safest method, both explained methods for the warning in pedestrian detection, obstacle box and pointcloud checking, were used together in the program.

3.13 Stop warning

An extra feature to the project is implemented during each iteration of the data processing. Every time new data arrives to the program from the .bag file, the pointcloud is checked, point by point recorded, in order to compare the distance of all those points to the vehicle. If such distance is smaller or equal than two meters a red "STOP" warning sign will appear at the top left corner of the image, indicating the driver to stop immediately due to a possible collision. 3.23

Figure 3.23. *Stop sign appears if any point in the pointcloud is closer than two meters to the vehicle.*

3.14 Warning sign detection

Another feature inside this project is to detect signs related to cargo, for example, flammable signs. To detect such signs in the image, several methods are possible using machine learning.

This process is developed in Matlab, and will be part of the loop in the final program in the future. Each iteration of the loop will get new image information from the video camera, each of those images will be analyzed separately in order to find the possible signs, if such sign appears in the image, a signal will be delivered to the driver or vehicle in order to be more careful at the time of manipulating the cargo.

3.14.1 Simple network with ImageLabeler

The first approach used to detect the flammable sign is to train a R-CNN neural network, for this, training images from the desired sign are used. To prepare those images, "image labeler" application in Matlab is used. 3.24

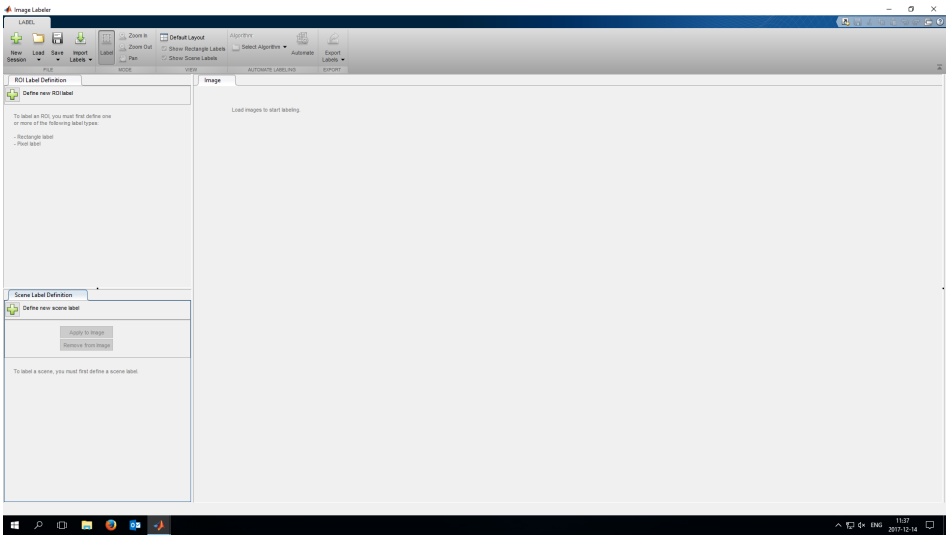

Figure 3.24. *Image Labeler window.*

The first step inside this application is to load all the training images inside its working space. Once they are loaded, one label will be given to the ROI that is wanted to be selected, this will give a proper name to the desired areas of the image, those areas will be used to train the neural network. The ROI will be selected manually as seen in figure 3.25,

Figure 3.25. *Ready to export session in Image Labeler.*

The ROI will indicate to Matlab the position of the sign that should be detected in new images, with such information, the neural network will know where to find the data to train itself. This ROI can be changed in the future if the training wants to be changed.

Once the ROI is selected in all the training images, the data from the image directory and the bounding box can be exported into a .mat file. This .mat file will be of value "groundTruth" and, inside it, there will be three different properties, "DataSource" in which the image directories will be stored, "LabelDefinitions" in which the appearing labels will be stored and "LabelData", containing all the corner points of the ROI in each image.

This file has all the information needed to train the neural network, but not in the appropriate format, to have it, a table will be created in Matlab's workspace with two variables, "imageFilename" indicating the directory of the image and "flammableSign" indicating the corner points of each ROI. The final table can be seen in figure 3.26.

22x2 table

	1 imageFilename	2 flammableSign
1	'\\intra.tut.fi\home\pascualo\My Documents\Desktop\imagesTFM\flamable\image001.jpg'	[220,167,158,188]
2	'\\intra.tut.fi\home\pascualo\My Documents\Desktop\imagesTFM\flamable\image002.jpg'	[153,109,94,122]
3	'\\intra.tut.fi\home\pascualo\My Documents\Desktop\imagesTFM\flamable\image003.jpg'	[120,77,105,133]
4	'\\intra.tut.fi\home\pascualo\My Documents\Desktop\imagesTFM\flamable\image004.jpg'	[122,82,246,333]
5	'\\intra.tut.fi\home\pascualo\My Documents\Desktop\imagesTFM\flamable\image005.jpg'	[85,87,55,77]
6	'\\intra.tut.fi\home\pascualo\My Documents\Desktop\imagesTFM\flamable\image006.jpg'	[22,31,81,94]
7	'\\intra.tut.fi\home\pascualo\My Documents\Desktop\imagesTFM\flamable\image007.jpg'	[258,100,276,324]

Figure 3.26. *Fields "ImageFileName" (left) and "flammableSigns" (right) inside table object.*

Now that the table has all the relevant information for the neural network, the program will start configuring it, in one of the experiments performed, a new neural network is used to be trained from scratch, the parameters used for it are:

```
layers = [ ...
    imageInputLayer([32 32 3], 'Name', 'inputimage')
    convolution2dLayer(5,20, 'Name', 'conv', 'NumChannels',
     ↪ 3, 'NumFilters', 32, 'Padding', [2, 2, 2, 2],
     ↪ 'BiasLearnRateFactor', 2)
    maxPooling2dLayer(3, 'Name', 'maxpool','Stride',2)
    reluLayer('Name', 'relu')
    convolution2dLayer(5,20, 'Name', 'conv_1',
     ↪ 'NumChannels', 32, 'NumFilters', 32, 'Padding', [2,
     ↪ 2, 2, 2], 'BiasLearnRateFactor', 2)
    reluLayer('Name', 'relu_1')
    averagePooling2dLayer(2, 'Name', 'avgpool', 'Stride',2)
    convolution2dLayer(5,20, 'Name', 'conv_2',
     ↪ 'NumChannels', 32, 'NumFilters', 64, 'Padding', [2,
     ↪ 2, 2, 2], 'BiasLearnRateFactor', 2)
    reluLayer('Name', 'relu_2')
```

```
11      averagePooling2dLayer(2, 'Name', 'avgpool_1',
          ↪ 'Stride',2)
12      fullyConnectedLayer(64,'Name', 'fc',
          ↪ 'BiasLearnRateFactor', 2)
13      reluLayer('Name', 'relu_3')
14      fullyConnectedLayer(2,'Name', 'fc_rcnn',
          ↪ 'WeightLearnRateFactor', 20,'BiasLearnRateFactor',
          ↪ 10, 'BiasL2Factor', 1)
15      softmaxLayer('Name', 'softmax')
16      classificationLayer('Name', 'classoutput')]
```

This will create a 15 layers neural network, such network will be trained by the program, based on the training images selected and the ROI inside each image.

The layers that forms the network perform the following tasks:

- ImageInputLayer, introduces the image in the network applying data normalization to it.
- Convolution2DLayer, uses sliding filters in the image in both axis, vertical and horizontal calculating the dot product of weights and image. Afterwards a bias term is added.
- MaxPooling2DLayer, downsamples the image into rectangular pooling regions and calculates the maximum in each one of those regions.
- ReLULayer, converts all the values smaller than zero in the image to zero.
- AveragePooling2DLayer, same as MaxPooling2DLayer but instead of calculating the maximum, calculates the average of each region.
- FullyconnectedLayer, multiplies the image by a weight matrix, when this process is done it also adds a bias vector.
- SoftmaxLayer, the image get a softmax function applied to it.
- ClassificationOutputLayer, gives the classification output of the network.

Apart from creating the neural network, also some options are given to maximize the accuracy of the training, those are:

```
1  options = trainingOptions('sgdm', ...
2    'MiniBatchSize', 32, ...
3    'InitialLearnRate', 1e-6, ...
4    'MaxEpochs', 10);
```

The parameters in the options indicate:

- MiniBatchSize, this parameter indicates the size of the mini-batch, being this, a part of the training set used to get the gradient of the loss function, having this, the weights will be changed.
- InitialLearnRate, this parameter indicates the speed of the training, its value can make the training extremely low or make it fast but giving worse results.
- MaxEpochs, determines the maximum number of epochs during the whole training.

With this, the training phase is ready to begin, to do so, "trainRCNNObjectDetector" function in Matlab is used, this will train our own neural network using the data already prepared. In our case, 10 epochs for training were used giving the result shown in figure 3.27

```
Training an R-CNN Object Detector for the following object classes:

* flammableSign

Step 1 of 3: Extracting region proposals from 22 training images...done.

Step 2 of 3: Training a neural network to classify objects in training data...

Training on single CPU.
Initializing image normalization.
|=========================================================================================|
|     Epoch    |   Iteration  | Time Elapsed |  Mini-batch  |  Mini-batch  | Base Learning|
|              |              |  (seconds)   |     Loss     |   Accuracy   |     Rate     |
|=========================================================================================|
|            1 |            1 |         0.77 |       0.6936 |       25.00% |     1.00e-06 |
|            4 |           50 |        26.63 |       0.6935 |       34.38% |     1.00e-06 |
|            8 |          100 |        54.06 |       0.6930 |       56.25% |     1.00e-06 |
|           10 |          140 |        74.93 |       0.6920 |       75.00% |     1.00e-06 |
|=========================================================================================|

Network training complete.

Step 3 of 3: Training bounding box regression models for each object class...100.00%...done.

R-CNN training complete.
```

Figure 3.27. *Training phase process.*

This process took in Matlab 90.90 seconds with this neural network, this time is a short time, as the number of training images is also very small, only 22 images were used for training, most probably, with a dataset of training images much bigger the detection, results could be better but the time needed to train the network would also be much longer. It has to be also taken into account that this training phase had only one class being trained in the classifier.

When this process ends, the neural network is already trained taking into account all the images and ROIs, with this trained network, new images can be used to test the detection; also, the training images can be validated, to check whether or not the training worked properly as seen in figure 3.28.

Figure 3.28. *Detection performed with trained network.*

As seen in figure 3.28, the flammable sign is properly detected in the image and marked as detection with the yellow rectangle, the string written on top of it indicates the label corresponding to the detection and its confidence, being this, how close is the detection to the real trained flammable sign.

One thing to take into account, as for the pedestrian detection part, is that resizing the image into a smaller one will reduce significantly the time taken for the detector to finish the process in that image, in this project, images are resized to 300x300 pixels.

During this process, the time needed to perform the detection inside a single image was 2.35 seconds, not good enough for real time video detection, but, it must be taken into account that, if the code is converted to C++, it will be much faster. Also all this tests were performed using CPU; future test using GPU will be performed and the time needed to perform the same opperations with a GPU with this kind of data will become much shorter.

3.14.2 Alexnet network with ImageLabeler

The second approach to perform the sign detection was to re-train a previously trained network, in our case, alexnet. This network already has one thousand labels and more than one million images that were used to train it.

To use such network, the first step is to load it inside Matlab's workspace. The difference with our previous network is that this one is already trained and that it has twenty five layers instead of fifteen, this can make it more reliable but will take much longer to train.

The process to use this network is the same as before, once the network is loaded, the options for the training will be selected, in this case they will be the same ones. The network will be re-trained for the new images and label, then it will be used to detect the warning sign in new images.

As in previous test, the network will then be used to detect the warning sign in new images. implemented in Matlab "trainRCNNObjectDetector" is used to re-train the network, in our case, with a set of images from flammable warning signs, the training was divided again in 10 epochs giving the result shown in figure 3.29.

```
Training an R-CNN Object Detector for the following object classes:

* flammableSign

Step 1 of 3: Extracting region proposals from 22 training images...done.

Step 2 of 3: Training a neural network to classify objects in training data...

Training on single CPU.
Initializing image normalization.
|=========================================================================================|
|     Epoch    |   Iteration  | Time Elapsed |  Mini-batch  |  Mini-batch  | Base Learning|
|              |              |  (seconds)   |     Loss     |   Accuracy   |     Rate     |
|=========================================================================================|
|            1 |            1 |        34.09 |       0.7889 |       62.50% |     1.00e-06 |
|            4 |           50 |      1672.67 |       0.1252 |       93.75% |     1.00e-06 |
|            8 |          100 |      3345.28 |       0.0974 |       93.75% |     1.00e-06 |
|           10 |          140 |      4678.34 |       0.1520 |       96.88% |     1.00e-06 |
|=========================================================================================|

Network training complete.

Step 3 of 3: Training bounding box regression models for each object class...100.00%...done.

R-CNN training complete.
```

Figure 3.29. *Re-training phase process for alexnet network.*

The time needed to perform the re-training was 4701.99 seconds, the same as 78.36 minutes or 1 hour and 18 minutes, the difference in time with the previous experiment is extremely big, but, in any case, with good results after training it should not be a problem, as the program does not need to train the network each time it starts.

Once the network is ready, it can be used to detect the flammable warning sign in new images. As seen in figure 3.30, even though the network is more complex the confidence is smaller and the bounding box of the detection is not perfectly selected, this might be because some tuning in the options is needed, or because such network is too complex for this task, also, it might be needed a bigger dataset of training images as the one used was only 22 images.

Figure 3.30. *Detection using Alexnet re-trained network.*

The time elapsed during the detection process for the image in figure 3.30 was 118.4 seconds or almost 2 minutes, this time is extremely long to perform real time detection, even when tunning the network. Some test using GPUs instead of CPUs can be done, in order to check if this scenario is repeated.

Having the big difference in time for both, re-train and detection of features in new images and having worse results when the detection is performed, this process was discarded from the project.

3.14.3 Alexnet network with imageDatastore object

To try to get some better results using alexnet network, the next test were performed using this network, again in Matlab.

The first step to perform this test, is to resize all the images to 227 x 227 pixels, as that size is needed to train alexnet network.

This time, to prepare the data used for training, an "imageDatastore" object is created, this object basically represent a collection of image files. The structure of such object contains the following fields:

- Files, this field contains in a vector of n x 1 being n the number of image files contained in the object. Each component forming this vector contains the directory where the image can be found in the computer.
- ReadSize, this field represents the number of files that will be read each time the read function is invoked.
- Labels, this field contains a vector of n x 1 positions, being n again the number of images contained in the object. Each of those position is the label given to the corresponding image in the same position inside "Files" field.
- ReadFcn, is the function in charge of reading image data taking as input the directory of the image to read.

Once the "imageDatastore" object is filled properly with the resized images, the whole dataset of images is divided into training images and validation images, being 70% and 30% of the original dataset respectively. Those divisions are again stored in two new "imageDatastore" objects.

The most important fields inside our "imageDatastore" objects are "Files" and "labels", those fields, when filled properly to train the neTtwork and validate its performance, will look as in figure 3.31.

1	\\intra.tut.fi\home\pascualo\My Documents\Desktop\imagesTFM\flamable_small\image006.jpg	1	flamable
2	\\intra.tut.fi\home\pascualo\My Documents\Desktop\imagesTFM\flamable_small\image007.jpg	2	flamable
3	\\intra.tut.fi\home\pascualo\My Documents\Desktop\imagesTFM\flamable_small\image008.jpg	3	flamable
4	\\intra.tut.fi\home\pascualo\My Documents\Desktop\imagesTFM\flamable_small\image011.jpg	4	flamable
5	\\intra.tut.fi\home\pascualo\My Documents\Desktop\imagesTFM\flamable_small\image013.jpg	5	flamable
6	\\intra.tut.fi\home\pascualo\My Documents\Desktop\imagesTFM\flamable_small\image016.jpg	6	flamable
7	\\intra.tut.fi\home\pascualo\My Documents\Desktop\imagesTFM\flamable_small\image018.jpg	7	flamable

Figure 3.31. *Fields "Files" (left) and "Labels" (right) inside validation images object.*

Once the data is ready to be used for training and validation, the program loads alexnet network and makes it ready for the training. Now, the last three layers need to be removed from the network in order for them to not interfere with the new training, in any case, those three layers are built again in the network before the training phase. As explained during previous test, alexnet network has 25 layers making it more complex than the one created for the first test. Those layers are the ones shown in figure 3.32.

There are two new layers with respect to the simplest network, those layers are:

- CrossChannelNormalizationLayers, provides a channel-wise local response normalization during the process.
- DropoutLayer, with a set probability, this layer might set the input elements to zero.

1	1x1 ImageInputLayer
2	1x1 Convolution2DLayer
3	1x1 ReLULayer
4	1x1 CrossChannelNormalizationLayer
5	1x1 MaxPooling2DLayer
6	1x1 Convolution2DLayer
7	1x1 ReLULayer
8	1x1 CrossChannelNormalizationLayer
9	1x1 MaxPooling2DLayer
10	1x1 Convolution2DLayer
11	1x1 ReLULayer
12	1x1 Convolution2DLayer
13	1x1 ReLULayer
14	1x1 Convolution2DLayer
15	1x1 ReLULayer
16	1x1 MaxPooling2DLayer
17	1x1 FullyConnectedLayer
18	1x1 ReLULayer
19	1x1 DropoutLayer
20	1x1 FullyConnectedLayer
21	1x1 ReLULayer
22	1x1 DropoutLayer
23	1x1 FullyConnectedLayer
24	1x1 SoftmaxLayer
25	1x1 ClassificationOutputLayer

Figure 3.32. *Layers inside alexnet network.*

Once the network is ready, some options need to be set in order to tune the training, in the case of this test the options are the following:

```
options = trainingOptions('sgdm',...
    'MiniBatchSize',10,...
    'MaxEpochs',4,...
    'InitialLearnRate',1e-4,...
    'Verbose',false,...
    'Plots','training-progress',...
    'ValidationData',validationImages,...
    'ValidationFrequency', 5;
```

The new options that appear during this method training phase are:

- Verbose, this option tells the program show or not the information of the training process in the command window.

- Plots, this parameter tells the program to show or not the plots of the training process.
- ValidationData, tells the program which variable stores the validation set to test the network.
- ValidationFrequency, indicates the amount of iterations taken between evaluations of validation metrics.

Once everything is set for the training, it can start, for that, the "imageDatastore" object with the training images, the network and the options are given to the function "trainNetwork" implemented in Matlab. This function will be the one in charge of the training of the network, giving the user real time feedback in the development of the training. When the training ends, this feedback window looks as shown in figure 3.33.

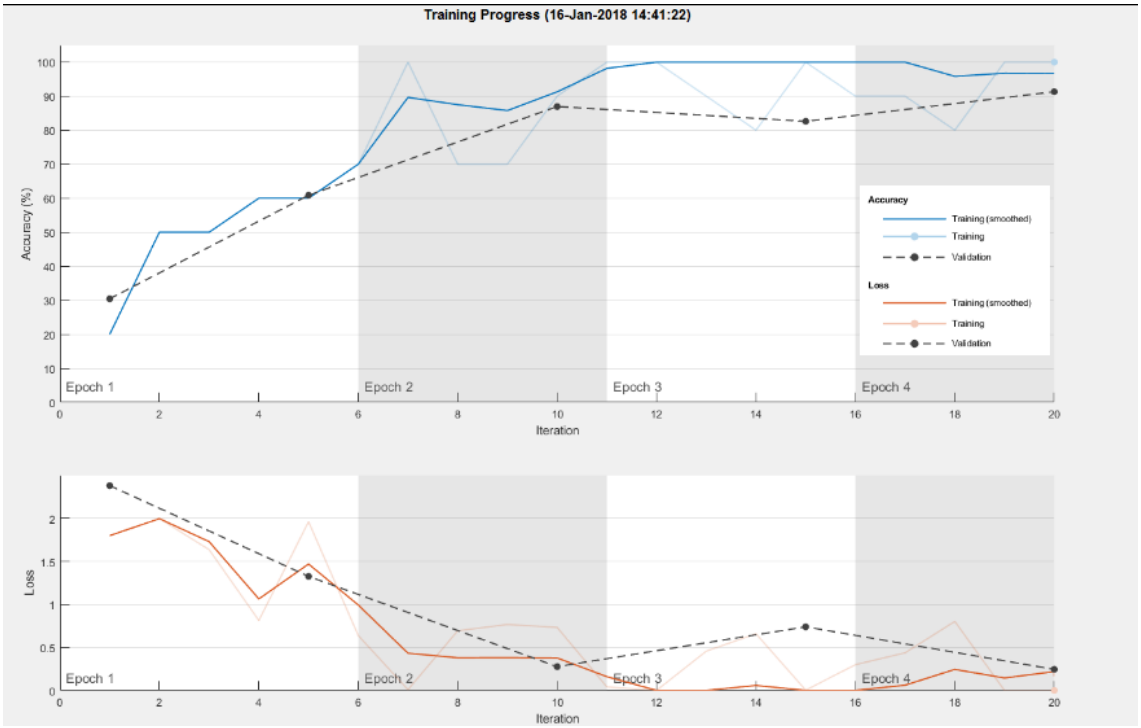

Figure 3.33. *Alexnet network training phase.*

Inside the plots of the training phase, the graphs that can be seen are:

- Training accuracy, is the accuracy obtained during each mini-batch.
- Training accuracy (smoothed), same as before but smoothing will help to detect trends during the training as it is less noisy.
- Validation accuracy, accuracy inside the validation set.
- Training loss, Cross entropy loss during the training.
- Training loss (smoothed), same as before but smoothed making the graph less noisy.
- Validation loss, loss on the validation set.

As seen at the right side of figure 3.33, the time needed to train this network, with 74 images from 4 different classes, is 4 minutes; is a bit longer than the best training time obtained until this test, but the set is almost four times bigger and has several classes instead of just one as in the previous tests.

The time needed to classify a single image using this trained network is 0.3 seconds, this is the best performance so far during the classification phase. Even though this classification time is far from real time video, this number will be significantly reduced when converting the code from Matlab to C++ and when using a GPU instead of a CPU, as in the case of this test, making possible the real time detection using this program.

To test the accuracy of the training, even though during the training phase the validation accuracy is 91.30%, the program calculates the validation accuracy also with the validation set of images; in this test, there were 23 validation images from all the different classes. Using such image set, the validation accuracy when classifying the images is 95.65%. The results of the classification can be seen in figure 3.34.

Figure 3.34. *Validation images classification results.*

Only one over twenty three images was classified wrong. As in the final program some mean between several frames would be made to avoid this kind of false classification, such small error rate can be neglected.

3.14.4 Googlenet network with imageDatastore object

One last test, to measure performance from different networks, time required for training and time required for the classification of images, were performed using googlenet network this time.

The process is pretty similar as the previous one using alexnet, but with some changes in the network and options.

The first step is to prepare the "imageDatastore" object with, again, all the image directories and the corresponding label for each image as seen in 3.31 during previous test explanation.

Once the "imageDatastore" object is filled properly with all the training and validation images, the images will be resized for them to be usable by googlenet network, this time the resize must be done to 224 x 224 pixels each image.

When the images have the correct size, the next step is to divide the whole image set into training and validation images, being 70% and 30% from the whole set respectively as during the previous test.

Again, to avoid problems during the training phase due to previous training of the network, the last three layers are removed and substituted by three new layers, this will make the new training to be performed in a smoother way, giving us better training time and classification results. Googlenet network will have 144 layers after this process, making it the most complex network during this project. The shape of the network can be seen in figure 3.35.

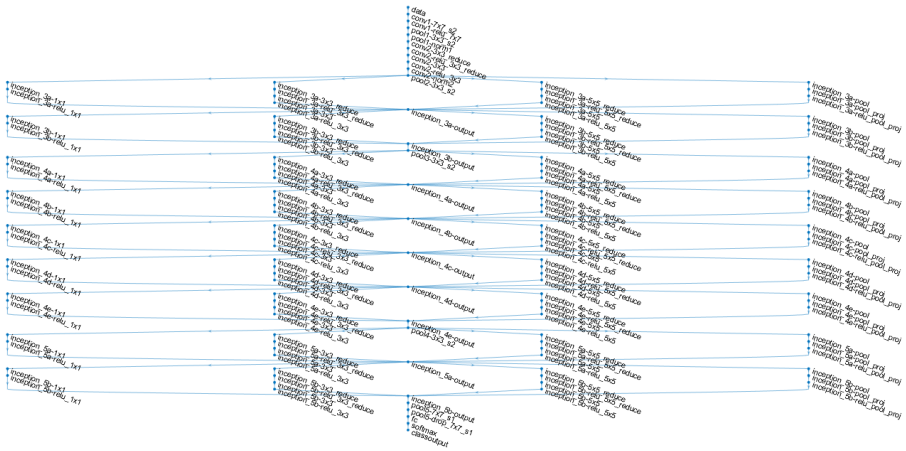

Figure 3.35. *Googlenet network layers.*

Once the network is ready for training, some options are tuned in order to obtain the best training result during the training phase. This tuning will give us better classification results during the validation phase. Those tunning options during this test are:

```
options = trainingOptions('sgdm',...
    'MiniBatchSize',10,...
    'MaxEpochs',3,...
    'InitialLearnRate',1e-4,...
    'VerboseFrequency',1,...
    'Plots','training-progress',...
    'ValidationData',validationImages,...
    'ValidationFrequency', 3;
```

During this test, there was real time feedback from the training phase as in the previous test, this makes easier to the programmer to understand what is going on during this phase. The final result of the training phase can be seen in figure 3.36.

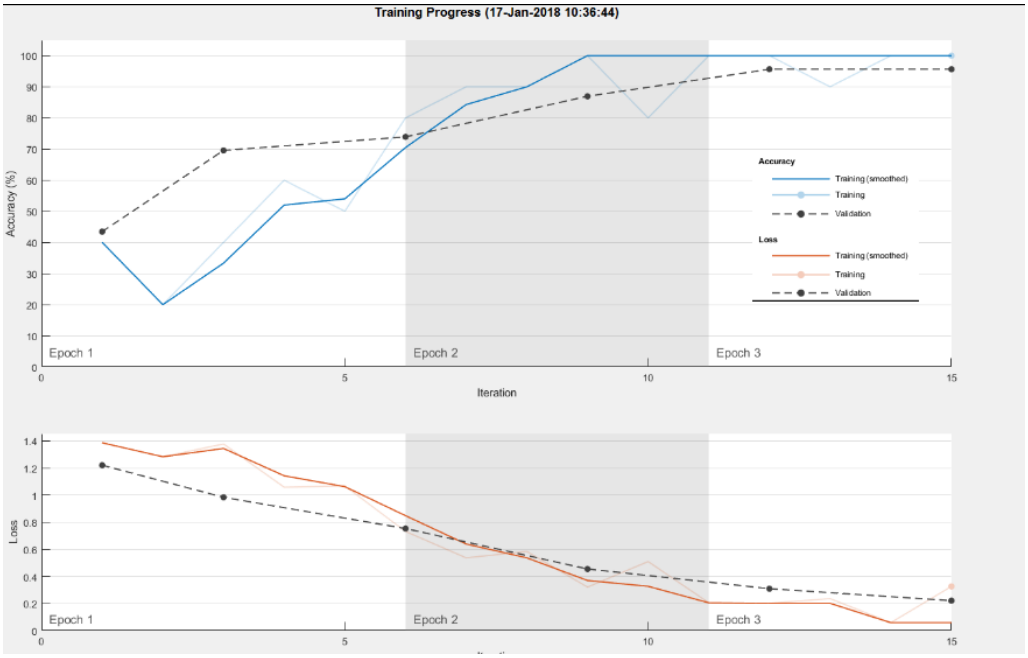

Figure 3.36. *Googlenet training phase.*

At the same time, some feedback about the training phase is given in the same way as during the first test to have the results of each iteration as time elapsed, validation loss or validation acuracy between some other data. 3.37

```
Training on single CPU.
Initializing image normalization.
|=========================================================================================================================|
|     Epoch    |   Iteration  | Time Elapsed |  Mini-batch  |  Validation  |  Mini-batch  |  Validation  | Base Learning|
|              |              |   (seconds)  |     Loss     |     Loss     |   Accuracy   |   Accuracy   |     Rate     |
|=========================================================================================================================|
|            1 |            1 |        26.79 |       1.3861 |       1.2205 |       40.00% |       43.48% |     1.00e-04 |
|            1 |            2 |        68.75 |       1.2829 |              |       20.00% |              |     1.00e-04 |
|            1 |            3 |        94.80 |       1.3772 |       0.9838 |       40.00% |       69.57% |     1.00e-04 |
|            1 |            4 |       135.89 |       1.0581 |              |       60.00% |              |     1.00e-04 |
|            1 |            5 |       161.88 |       1.0680 |              |       50.00% |              |     1.00e-04 |
|            2 |            6 |       187.61 |       0.7320 |       0.7533 |       80.00% |       73.91% |     1.00e-04 |
|            2 |            7 |       228.93 |       0.5369 |              |       90.00% |              |     1.00e-04 |
|            2 |            8 |       254.88 |       0.5844 |              |       90.00% |              |     1.00e-04 |
|            2 |            9 |       281.51 |       0.3210 |       0.4549 |      100.00% |       86.96% |     1.00e-04 |
|            2 |           10 |       323.08 |       0.5096 |              |       80.00% |              |     1.00e-04 |
|            3 |           11 |       349.76 |       0.2052 |              |      100.00% |              |     1.00e-04 |
|            3 |           12 |       376.52 |       0.2014 |       0.3093 |      100.00% |       95.65% |     1.00e-04 |
|            3 |           13 |       418.27 |       0.2359 |              |       90.00% |              |     1.00e-04 |
|            3 |           14 |       448.08 |       0.0586 |              |      100.00% |              |     1.00e-04 |
|            3 |           15 |       476.16 |       0.3261 |       0.2212 |      100.00% |       95.65% |     1.00e-04 |
|=========================================================================================================================|
Elapsed time is 498.797531 seconds.
```

Figure 3.37. *Googlenet training phase results.*

For this network, having the same amount of images and classes than in previous test, the time needed for training was of 498.79 seconds or 7 minutes and 56 seconds, almost double of the time needed for alexnet network. The time needed to classify a single image is 0.95 seconds, almost three times larger than with alexnet network.

In any case, this value of classification time might be good enough for real time classification of images in video, after coverting the code from Matlab to C++ and while making the computations with a GPU instead of a CPU.

The final value that is interesting in this method is the validation accuracy, being in this test 95.65%. Only one image out of 23 inside the validation image set was classified wrong, this gives us the same accuracy than with alexnet network during the previous test.

Once the training ends, the validation accuracy can be tested on the validation images that were separated from the whole set at the beggining of the test. Each of the 23 images inside this validation test will be classified one by one by the trained network in order to check the results. Those results are shown in figure 3.38.

Figure 3.38. *Googlenet validation results.*

4. RESULTS AND ANALYSIS

During this chapter, the results obtained in each test performed during the project will be shown after a brief explanation of the test itself, to give the reader a wider view of the problem and its solution.

4.1 Main test environment

One of the tests performed for data recording, consisted into placing the illumination unit, the 3D sensor and the video camera on the back bumper on the vehicle, this way, the objects behind it would be visible much sooner, giving the vehicle or the driver more time to maneuver in order to place the vehicle in the most appropriate position to proceed to the pick up of the demountable.

One of the main fears, was the possibility of the cargo or some moving part of the vehicle, to block the view of the sensors giving fake data that would interfere with the way the vehicle is supposed to move and position, but, after the first test, the team realized that the sensors were placed properly, giving no error in the data at the time to move the vehicle.

The only possible problem using this position is that, when the vehicle is very close to the demountable, 1-2 meters, the video camera stop viewing the hitch, at this distance, the vehicle should be positioned properly, but seeing the hitch at any time would create additional useful data for perfect positioning and pick up. The last meter before reaching the demountable can be seen in figure 4.1.

Figure 4.1. *View of video camera at 1 meter of the demountable.*

The team considered the results from this test successful, as there was no problem when recording data and the distance needed to detect the demountable decreased by the length of the vehicle basically. The data recorded was very useful at the time to check the results and how the program was performing.

4.2 Pedestrian detection and warning

The process performed to achieve pedestrian detection is the same as explained in section 3.12.3. Once having the appropriate calibrated .launch file, the program will perform the appropriate conversions in the data when required. Also, the downsize of the image was required to achieve real time pedestrian detection. This detection be seen in figure 4.2.

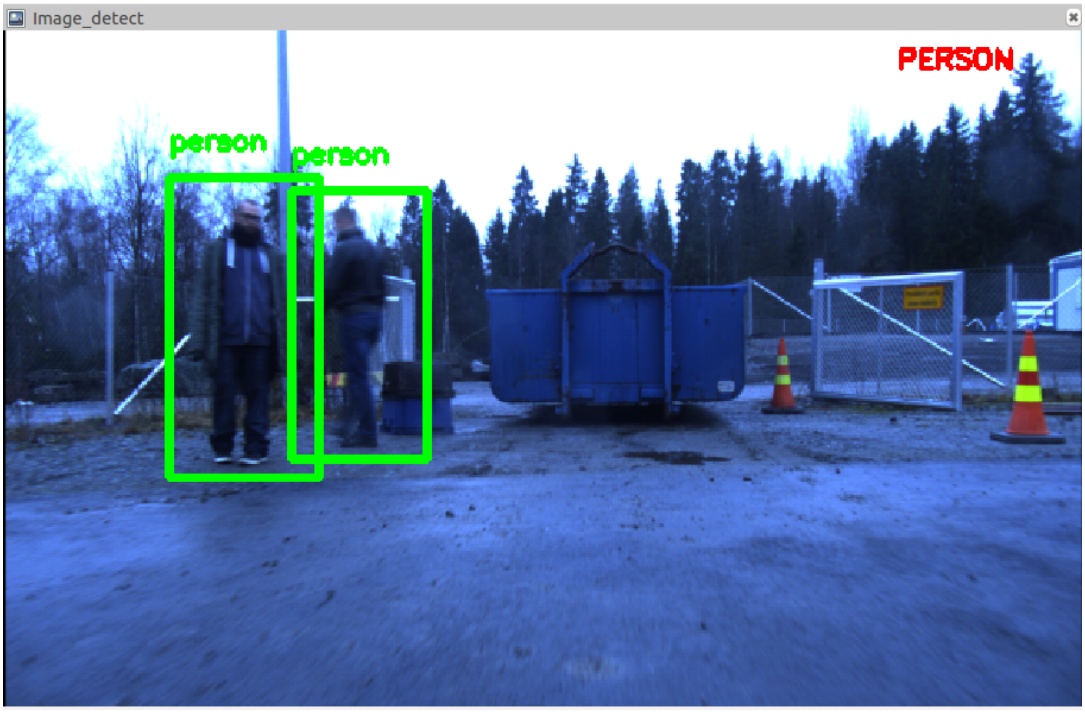

Figure 4.2. *Pedestrian detection in the recorded image.*

The 3D data recorded will be at the same time showed as a point cloud in the 3D space created in Rviz. Rviz will let the user navigate freely through all the 3D positions, this way he or she can watch the results in the way is needed. The grid that appears in this space will be divisions of 1 meter x 1 meter. Also, in this representation, a green box is used to represent the position of the demountable, having also an arrow indicating the normal of the closest face to the vehicle placed in the position where the hitch is supposed to be. All this can be seen in figure 4.3.

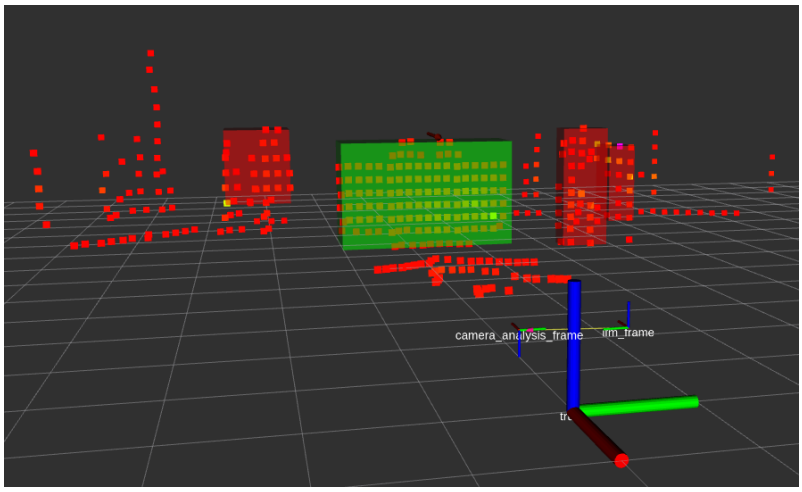

Figure 4.3. *Rviz representation of 3D space recorded.*

Having the position of the demountable in 3D, the position comparison with respect to the detected pedestrians can be done. This process is explained in section 3.12.3.

While all this process is being computed, also the region of interest of the image, the demountable, is cut from it, so it could be processed to find, for example, possible warning signs or company logos, in order for the vehicle to proceed accordingly to the pick up.

The results from this process were considered successful, as there were almost, or even zero, false positives during the pedestrian detection after the tunning of the detector. The detection could be better in some conditions, as not always there was a person detected in the image when there was one as, sometimes, the image became blurry due to the movement. In any case, the results satisfied the team during the project performance.

A minor change was needed in one test, as the video camera was placed in the vehicle upside down because of how the box containing it had the screws placed. The upside down image can be seen in figure 4.4.

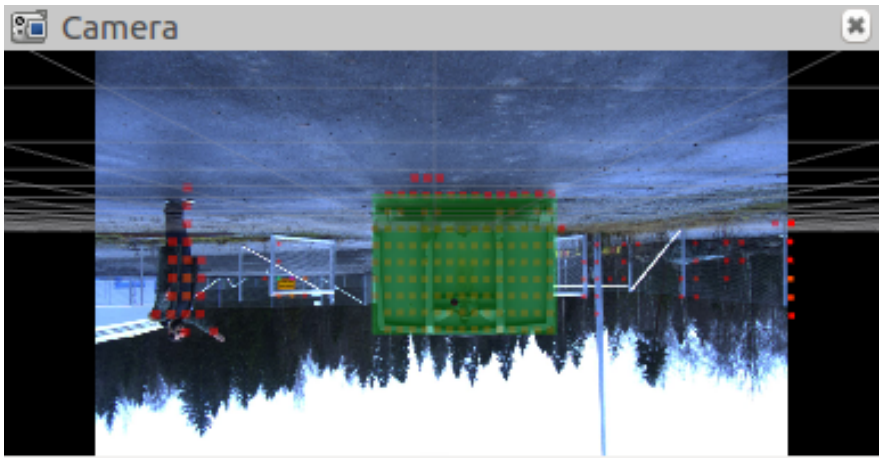

Figure 4.4. *Flipped image with real time point cloud positioning.*

The solution for this problem was, basically, change the .launch file in ROS so when the conversion of the point cloud and ground level to the camera frame was made, it would position the data properly in the image, as seen in figure 4.4.

In figure 4.4, it can be also observed that the ground plane is also shown in the point cloud positioning, this will help also in calculations reducing the time needed to compute the position of the pixels, avoiding irrelevant data.

At the same time, to flip the real time video image was needed in order to perform the pedestrian detection, because, when the program is performing the look up for HOG features representing human beings, it only does such search as if they where standing up, therefore, a tilt in the recording angle would give no positive results when performing the pedestrian search in the image. The solution for this was very simple, the image was flipped to the appropriate position. At the same time it had to be downsized to mate the frame by frame pedestrian detection process faster, in order to obtain real time detection in the final program. This result can be seen in figure 4.5.

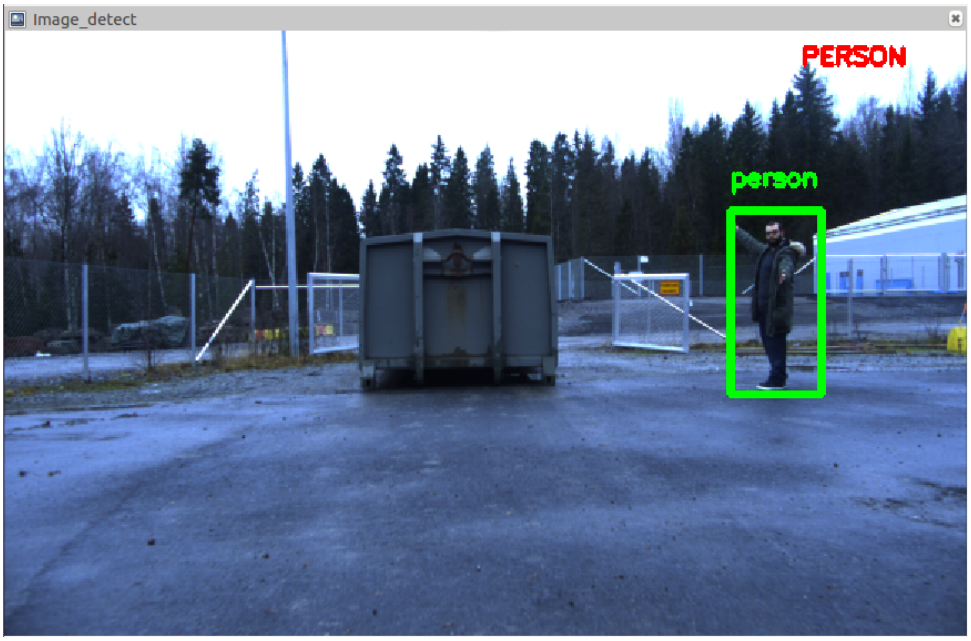

Figure 4.5. *Flipped image to perform pedestrian detection.*

Apart from those minor changes, the process is the same as explained in section 3.12.3.

4.2.1 Image recorded in normal light conditions.

Most of the tests were recorded during day conditions, so, enough light appeared in the videos without needing extra lights in the recording process. In all videos, the recording was made between 12:00 and 13:00 in November. This meant enough light for the camera to record without any problem, even though in all the recordings the sky was very cloudy. These conditions can be seen in figure 4.6.

Figure 4.6. *Regular light conditions during recordings.*

The recordings obtained in such light conditions performed perfectly during the testing of the C++ program.

4.2.2 Image recorded with high intensity light conditions.

One test was recorded having the back lights from the vehicle on. Those lights are supposed to be strong enough to maybe affect the recorded image, for example, making it too bright in some parts, making the detection of the signs, the demountable or the pedestrians impossible.

After processing the data and checking its performance inside the C++ program, the team discovered that the lights did not affect at all the image, making them perfect to be used in this kind of system. How the image looked like after recording using such strong lights is shown in figure 4.7.

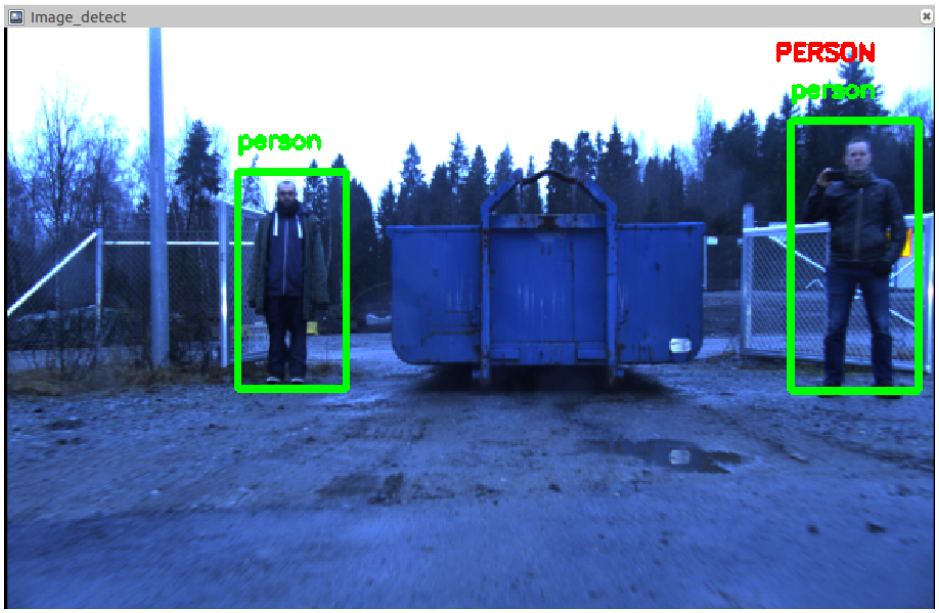

Figure 4.7. *Recording with high intensity light conditions.*

As seen in figure 4.7, this data performed exactly as the ones recorded without this kind of lights, so, it was considered perfectly valid for future test and no changes in the lights or in the devices was needed for the project to continue, as all the detections were performed properly.

4.2.3 Pedestrian detection in Matlab

Two extra tests were performed during the project to obtain pedestrian detection using Matlab.

During the first, motion-based object detection was performed in real time. This option was quickly discarded, as to use a motion based system in a moving vehicle would not give the desired results in the final program. The method to perform this test is explained in subsection 3.12.1.

The second test was performed using Aggregate Channel Features to perform the pedestrian detection in Matlab. This process did give good results with the real time pedestrian detection but, as the project was mainly implemented in C++ at the moment of this test and the project already used OpenCV libraries, this method was discarded in order to try to join as much as possible the code into the same project file in C++. How this test was performed is explained during subsection 3.12.2.

4.3 Obstacle warning

Inside this project, there is also implemented at the same time of pedestrian detection, obstacle detection and warning, in case any obstacle recorded by the 3D sensor could represent a problem for the vehicle when approaching the demountable to pick it up.

This method obtains the position of the obstacles already preprocessed. Once such information is obtained, the program will compare the position of the corners of those obstacles in the horizontal axis with the position of the corners of the container. Left corner of the demountable with right corner of the obstacles and right corner of the demountable with left corner of the obstacles. If such positions are further away than 0.5 meters, the image will show those obstacles in green, if they are closer, the obstacle position will be marked in red, advising the driver or the vehicle to stop immediately and move such problematic obstacle(s).

In figure 4.8, it can be seen that, the boxes at the left did not fulfill the obstacle safety measures and, therefore, were marked as red while the other obstacles did stay in green, in this case, because they were placed behind the front of the demountable. All safety measures are explained in section 3.10.

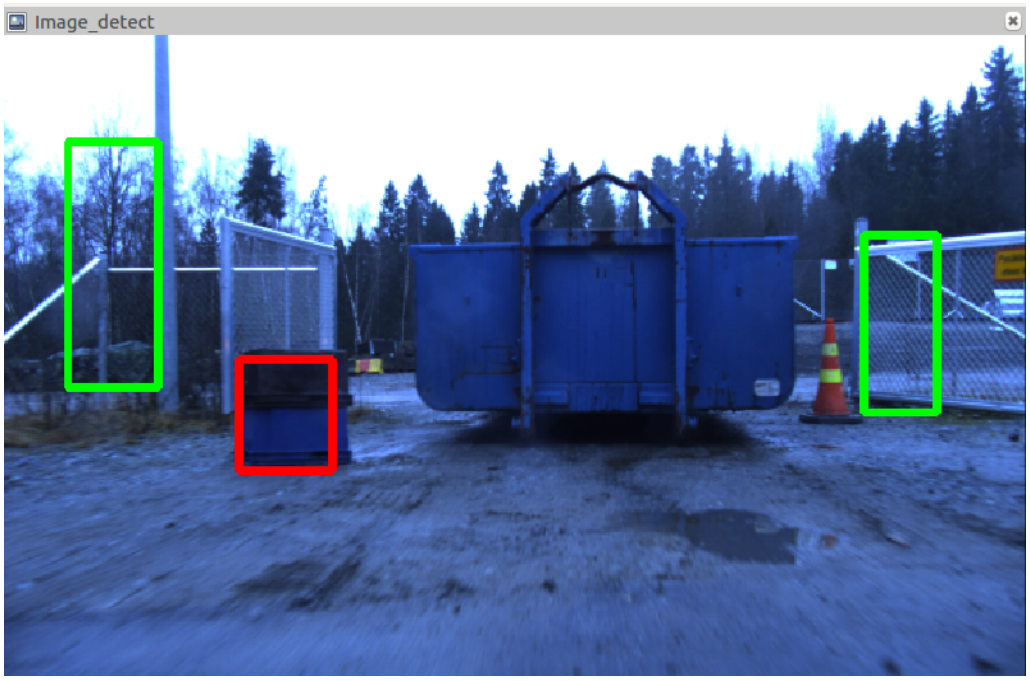

Figure 4.8. *Obstacle warning.*

The results of this process were successful in all conditions, having a variable in the code to quickly change the minimum allowed distance to the demountable if needed for some other projects.

4.4 Warning signs detection

This part of the project have been only implemented in Matlab, after checking that this codes can be easily converted into C++ codes and merge them with the main code file. Basically, machine learning methods were trained to detect several warning signs as flammable, explosive or biohazard, as well as the logo from TUT, considering it the logo from a company that could be drawn in the demountable. This detection will help when picking up the cargo for the driver or vehicle to know when to handle it with more care or which demountable should pick up. For this part of the project several methods were tested.

4.4.1 Simple network with ImageLabeler

During this test, a simple network of 15 layers was trained using only the region of interest from 22 images from the warning sign "flammable". The results were satisfying, knowing that only one class was trained in the classifier and knowing the times needed for training (90.90 seconds), but the classification time of the images (2.35 seconds) was too long for real time applications.

Other counter part from this method is that, some parts of the images, sometimes, were classified as flammable sign, also while they were not (false positives). To try to avoid this happening, a more complex neural network, alexnet, was trained instead. This network was trained following the method explained in subsection 3.14.1.

4.4.2 Alexnet network with ImageLabeler

During this test, the same method as with the previous one was used but, in this case, the network trained was alexnet network, having 25 layers instead of 15, making it more complex to train. This statement could be visibly measured when performing the training, as it took 1 hour and 18 minutes to train, a very much longer time than in previous case. This could not even be a problem as, with a working network in the classification phase, there is no need for more training, but the real problem appeared while in this phase almost 2 minutes were needed to classify a single image. Besides that, the classification results were a bit worse than in previous cases, having the bounding box of the classified signal usually bigger than the actual sign. This network was trained using the method as explained in subsection 3.14.2.

For this reason this method was discarded, but alexnet was a very good option in theory, so another solution was prepared using this network.

4.4.3 Alexnet network with imageDatastore object

To test again alexnet network performance, this network was used but trained in a different way, instead of using the region of interest in the training images, the whole image was used to train the network. Having the image sizes reduced to 227 x 277 pixels, this training phase lasted 4 minutes while using 74 images from 4 different classes instead of 22 images from a single class, as in previous test. This time is much better for training, letting the developer to try different options much faster and see the results.

At the same time, the classification of a single image was performed in 0.3 seconds, this amount of time, after the conversion to C++ code and when using a GPU for classification, will be reduced, making this method the best one so far for the project.

The best accuracy obtained during this method was of 95.65% correct validation, making only one mistake out of 23 validation images. The method to train the network is explained in subsection 3.14.3.

After considering the results from all the different tests, validation of images in a recorded video was tested using this method, as it was the one with the best results. This test gave very good results, giving a very high accuracy while recording different warning signs plus the TUT logo printed in regular A4 paper. The video was recorded with the camera of a mobile phone.

The recorded videos tried to obtain the images straight and with some sort of incorrect positioning, moving the phone in different angles, to try to obtain possible positions that might occur when approaching the demountable with the vehicle.

During this video test, almost real time classification was achieved, making the team think again that validation using a GPU will obtain real time classification without problem.

4.4.4 Googlenet network with imageDatastore object

One last test was performed to compare the performance of alexnet network with googlenet network. Googlenet is the most complex network used in the project having 144 layers. The method to train this network is the same as during the previous test, as explained in subsection 3.14.4.

Basically, this test is the same as the previous one, just changing the network to googlenet and changing the image size to 224 x 244 pixels. The difference in results with previous test were that, the training phase lasted almost 8 minutes and the classification time for a single image took 0.95 seconds. Those times are worse than in previous test but again, the training time is not a big problem once the network is trained and the classification time could also achieve real time classification, when converting the code to C++ and classifying using a GPU.

The accuracy obtained for this method was, again, 95.65% giving us only one mistaken classification over 23 validation images, making this method the second best inside this project, only for the time needed to train and classify.

4.4.5 Results comparison

For the viewer to visualize better the results from each test, the table 4.1 shows all the results obtained during the sign classification process, marking as green the best result in each category and in blue the best performance classifier. Comparing all those results, the best performance detector for the warning signs and company logos was alexnet using imageDatastore objects for training it.

There must be taken into account that, the results shown in the table 4.1 are the most common results during the project but, in some particular cases, 100% accuracy was achieved during training and validation phases for both, alexnet and googlenet networks.

Table 4.1. *Warning sign and company logo detection results*

Method-Result	Training time (seconds)	Validation time (seconds)	Accuracy (%)
Simple network ImageLabeler	90.90	2.35	-
Alexnet ImageLabeler	4701	118.4	-
Alexnet objectDatastore	239.8	0.3	95.65
Googlenet objectDatastore	498.79	0.95	95.65

The final test to check the performance of the program was to classify the three warning sings plus the TUT logo in a previously recorded video. The results for this test gave almost real time classification, making the team think again that real time classification could be achieved using GPU for the classification task; also, highg accuracy during the classification of the images in the frames of the video was achieved, proving the accuracy of this method. The videos were recorded using the camera of a mobile phone, pointing to printed signs and TUT logo on regular A4 paper, changing from one print to another to test the 4 four possible classes, and also, moving the camera in different angles to obtain more real "not ideal" images that will approximate more to the possible ones when approaching the demountable with the vehicle. Some classified frames can be seen on figure 4.9.

Figure 4.9. *Validation results from pre-recorded video.*

5. CONCLUSION

During this chapter, the conclusions made during the development of the project will be analyzed, taking into account the process and results obtained. Also, future modifications and improvements will be written at the end of it.

5.1 Conclusion

The main goal of the project was to perform safety analysis based on neural networks, in order to achieve extra information while a cargo vehicle is moving. This safety measures were divided into pedestrian detection and warning, obstacle avoidance warning, proximity warning to obstacles, warning sign classification and company logo classification.

To perform those processes, several programs were created in C++ and Matlab. The program in C++, will be the one in charge of the pedestrian detection, obstacle avoidance warning (including pedestrians) and proximity warning.

Pedestrian detection part will detect in real time, from the recorded image of a video camera, the position of the possible pedestrians in the image, having the less possible false positives and false negatives while the process is performed.

Obstacle avoidance warning will perform the necessary computations, as coordinate transform of the data obtained by the 3D sensor and comparison between the positions of the demountable and the obstacles. If an obstacle is closer than 0.5 meters to the demountable, or a pedestrian is closer than 2m to it, a warning sign will appear in order for the driver or the vehicle to stop.

The proximity warning will analyze the position of every single point inside the 1024 point grid recorded by the 3D sensor in real time, if any of those points is less than 2 meters close to the vehicle, an immediate warning signal is given to the driver or vehicle to stop.

The Matlab part of the project is in charge of the warning signs and company logo classification. Several neural networks were trained using an image dataset of the signs and logos to classify. The best result was given with alexnet network, being this the second fastest to train, the fastest in image validation and the best with googlenet in accuracy.

The main problems during this process were that, the image dataset was not big enough to, maybe, get better results in the classification part of the warning signs and that the video camera could have better resolution and performance while in movement, as sometimes the image becomes blurry and the pedestrian detection is more difficult to perform. In any case, the results in both parts were satisfying and the team considered such results successful.

5.2 Future work

At the moment, this is a "work in progress" project and, as explained during the objectives part during the introduction chapter, several tasks will be implemented in the future, in order to obtain fully automation for the vehicle and pick up of the cargo.

In this section, some of the possible future lines of work, modifications or improvements that have been coming to the mind of the team and the writer during the whole project, will be enumerated and briefly explained:

- Increase the image dataset to train the networks, an increase in the number of images used to train the network, having different light conditions or images taken from different angles, could increase the accuracy while detecting the different warning signs, making this program more trustworthy by the time of implementing it in a real product.
- Separate networks for warning sign detection and company logo detection, during this project, warning signs and logos were placed in the same network, as only one logo (TUT logo) was used in the classification, due to license problems. In the future, several logos and even more warning signs could be trained in two different networks to avoid wrong classification during the process.
- In the same way, train a network as, for example, alexnet, with a dataset of images for pedestrian detection in a related environment in which the vehicle will be performing its tasks, for example, instead of only detecting shapes using HOG features, also detecting the clothes that the employees inside, for example, a warehouse should be wearing.
- Extra obstacles detection. Sometimes there could be other moving obstacles in the image as dogs or cats, to train the network to also detect such animals will help making the system safer in the final product.
- Upgrade equipment to perform the classification of the different warning signs and logos using a GPU instead of a CPU, this could also be used during the pedestrian detection part.
- Increase 3D sensor resolution. The 3D sensor used during the project had a resolution of 16 x 64 points, the resolution in this case was not the best one while recording data, as some of the obstacles were difficult to detect. Upgrade the 3D sensor to a solid state LIDAR with better resolution, or attaching such LIDAR to a servo capable of moving, at least in the vertical axis as this is the one with less resolution, could help to create a more dense pointcloud, making the detection in it much more trustworthy.
- Automatize the pick up of the cargo, if the program is capable of detecting the hitch in the demountables, the vehicle could automatically pick it up using some sort of robotic arm.

- Automation of vehicle movement. The long therm objective of the project is to fully automate the vehicle for the pick up, this automation will include self driven vehicle, this could be achieved making the C++ program give the indicated signals as the speed permitted when approaching the demountable to the vehicle, to stop if some obstacle is blocking the way, to move slower if a pedestrian is detected in the image, or how to perform the pick up of the demountable if it contains dangerous materials.

REFERENCES

[1] "rqt_graph," ROS collaboration, 2014. [Online]. Available: http://www.ros.org/rqt_graph

[2] T. Linderberg, "Edge detection and ridge detection with automatic scale selection," *International journal of Computer Vision*, vol. 30, no. 2, pp. 117–154, 1998.

[3] "Benjamin tyner," Purdue University, 2006. [Online]. Available: http://www.stat.purdue.edu/~btyner/packages.html

[4] "Product o3m950," IFM, 2014. [Online]. Available: http://www.ifm.com/ie/en/product/O3M950

[5] "Product o3m250," IFM, 2016. [Online]. Available: http://www.ifm.com/ie/en/product/O3M250

[6] "Ui-5260cp rev. 2," IDS, 2017. [Online]. Available: https://en.ids-imaging.com/store/ui-5260cp-rev-2.html

[7] T. Ringbeck and B. Hagebeuker, "A 3d time of flight camera for object detection," 2007.

[8] M. Heredia Conde, *Compressive Sensing for the Photonic Mixer Device: Fundamentals, Methods and Results*. Springer Fachmedien Wiesbaden GmbH, 2017.

[9] "About ros," ROS collaboration, 2017. [Online]. Available: http://www.ros.org/about-ros/

[10] "Ros / introduction," ROS collaboration, 2014. [Online]. Available: http://www.ros.org/ROS/introduction

[11] "Navigating the ros filesystem," ROS collaboration, 2015. [Online]. Available: http://www.ros.org/ROS/tutorials/NavigatignTheFileSystem

[12] "Conceptual overview of catkin," ROS collaboration, 2017. [Online]. Available: http://www.ros.org/catkin/conceptual_overview

[13] "Nodes," ROS collaboration, 2012. [Online]. Available: http://www.ros.org/Nodes

[14] "Messages," ROS collaboration, 2016. [Online]. Available: http://www.ros.org/Messages

[15] "Topics," ROS collaboration, 2014. [Online]. Available: http://www.ros.org/Topics

[16] "Bags," ROS collaboration, 2015. [Online]. Available: http://www.ros.org/Bags

[17] "roscore," ROS collaboration, 2016. [Online]. Available: http://www.ros.org/roscore

[18] S. E. Umbaugh, *Digital image processing and analysis: human and computer vision applications with CVIPtools (2nd ed.)*. CRC Press, 2010.

[19] H. Barrow and J. Tenenbaum, "Interpreting line drawings as three-dimensional surfaces," no. 17, pp. 75–116, 1981.

[20] W. Zang and F.Bergholm, "Multi-scale blur estimation and edge type classification for scene analysis," *International journal of Computer Vision*, vol. 24, no. 3, pp. 219–250, 1997.

[21] D. Ziou and S. Tabbone, "Edge detection techniques: An overview," *International journal of Pattern Recognition and Image Analysis*, vol. 8, no. 4, pp. 537–559, 1998.

[22] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.

[23] I. Biederman, "Recognition-by-components: A theory of human image understanding," *Psychological review*, vol. 94, no. 2, pp. 115–147, 1987.

[24] J. L. Bentley, "Multidimensional binary search trees used for associative searching." *Communications of the ACM*, vol. 18, no. 9, p. 509, 1975.

[25] A. Fusiello, "Elements of geometric computer vision," University of Edinburgh, 2013. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/FUSIELLO4/tutorial.html

[26] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection." *Computer Vision and Pattern Recognition.*, 2005.

[27] C. Cortes and V. Vapnik, "Support-vector networks." *Machine Learning.*, vol. 20, no. 3, pp. 273–297, 1995.

[28] "De costa a costa sin conductor," EL MUNDO, 2015. [Online]. Available: https://www.elmundo.es/motor/2015/03/17/550812f7e2704e5d4f8b4585.html

[29] "The machine learning algorithms used in self-driving cars," KDnuggets, 2017. [Online]. Available: https://www.kdnuggets.com/2017/06/machine-learning-algorithms-used-self-driving-cars.html

[30] "Qué es un lidar y cómo funciona el sensor más caro de los coches autónomos," Motorpasión blog, 2017.

[Online]. Available: http://www.motorpasion.com/tecnologia/que-es-un-licar-y-como-funciona-el-sistema-de-medicion-y-deteccion-de-objetos-mediante-laser

[31] "Lidar vs radar in autonomous vehicles," Digi-Key electronics, 2017. [Online]. Available: http://www.digikey.caom/en/blog/lidar-vs-radar-in-autonomous-vehicles

[32] "Leddar technology fundamentals," LeddarTech, 2018. [Online]. Available: https://leddartech.com/technology-fundamentals/

[33] C. H. Chen, *Emerging Topics in Computer Vision and Its Applications*. World Scientific, 2012.

[34] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning (Adaptive Computation and Machine Learning series)*. Teh MIT Press, 2012.

[35] M. Jordan, *Neural Networks*. In Allen B. Tucker, 2004.

[36] "Statistics and machine learning toolbox," Mathworks webpage. [Online]. Available: https://www.mathworks.com/products/statistics.html

[37] "Image processing toolbox," Mathworks webpage. [Online]. Available: https://se.mathworks.com/products/image.html

[38] "Computer vision system toolbox," Mathworks webpage. [Online]. Available: https://se.mathworks.com/products/computer-vision.html

[39] "Matlab coder," Mathworks webpage. [Online]. Available: https://se.mathworks.com/products/matlab-coder.html

[40] "Motion-based multiple object tracking," MathWorks, 2018. [Online]. Available: https://se.mathworks.com/help/vision/examples/motion-based-multiple-object-tracking.html

[41] "peopledetectoracf," MathWorks, 2018. [Online]. Available: https://se.mathworks.com/help/vision/ref/peopledetectoracf.html

[42] "Object detection," OpenCV 2.4.13.4 documentation, 2017. [Online]. Available: https://docs.opencv.org/2.4/modules/gpu/doc/object_detection.html